Basic notions of algorithm theory

Peter Hajnal

Bolyai Institute, TTIK, University of Szeged, Hungary

Fall semester, 2023

The "definition" of algorithm

The "definition" of algorithm

- The algorithmic solution of a computational problem is a process, that for any given data performs a well-defined sequence of elementary steps and compute the corresponding answer.

The "definition" of algorithm

- The algorithmic solution of a computational problem is a process, that for any given data performs a well-defined sequence of elementary steps and compute the corresponding answer. Every elementary step is an obvious operation that can be executed by everyone.

The "definition" of algorithm

• The algorithmic solution of a computational problem is a process, that for any given data performs a well-defined sequence of elementary steps and compute the corresponding answer. Every elementary step is an obvious operation that can be executed by everyone.

• In order to specify an algorithmic problem we need to describe the following "components":

The "definition" of algorithm

• The algorithmic solution of a computational problem is a process, that for any given data performs a well-defined sequence of elementary steps and compute the corresponding answer. Every elementary step is an obvious operation that can be executed by everyone.

• In order to specify an algorithmic problem we need to describe the following "components":

(1) Computational problem (data/answer relation),

The "definition" of algorithm

• The algorithmic solution of a computational problem is a process, that for any given data performs a well-defined sequence of elementary steps and compute the corresponding answer. Every elementary step is an obvious operation that can be executed by everyone.

• In order to specify an algorithmic problem we need to describe the following "components":

(1) Computational problem (data/answer relation),

(2) Set of elementary steps.

The "definition" of algorithm

• The algorithmic solution of a computational problem is a process, that for any given data performs a well-defined sequence of elementary steps and compute the corresponding answer. Every elementary step is an obvious operation that can be executed by everyone.

• In order to specify an algorithmic problem we need to describe the following "components":

(1) Computational problem (data/answer relation),

(2) Set of elementary steps.

• The above description is a naive definition.

The "definition" of algorithm

- The algorithmic solution of a computational problem is a process, that for any given data performs a well-defined sequence of elementary steps and compute the corresponding answer. Every elementary step is an obvious operation that can be executed by everyone.

- In order to specify an algorithmic problem we need to describe the following "components":

(1) Computational problem (data/answer relation),

(2) Set of elementary steps.

- The above description is a naive definition. The word naive, used in an attributive form, means that we give up mathematical precision.

The language of algorithm theory

The language of algorithm theory

- As studying algorithm theory we need to learn the language of algorithms.

The language of algorithm theory

- As studying algorithm theory we need to learn the language of algorithms.

- The given data is called *input*.

The language of algorithm theory

- As studying algorithm theory we need to learn the language of algorithms.
- The given data is called *input*.
- The computed answer is called *output*.

The language of algorithm theory

- As studying algorithm theory we need to learn the language of algorithms.

- The given data is called *input*.

- The computed answer is called *output*.

- If we are given an input, say ω and an algorithm, while performing the described elementary operations we say that we *run the algorithm on input* ω.

Algorithms from school

Algorithms from school

- Performing multiplication, or addition is running a simple algorithm.

Algorithms from school

• Performing multiplication, or addition is running a simple algorithm. The elementary steps are the operations on digits.

Algorithms from school

- Performing multiplication, or addition is running a simple algorithm. The elementary steps are the operations on digits.

- Finding the greatest common divisor of given two numbers is a computational task.

Algorithms from school

• Performing multiplication, or addition is running a simple algorithm. The elementary steps are the operations on digits.

• Finding the greatest common divisor of given two numbers is a computational task. The Euclidean algorithm should be known from BSc.

Algorithms from school

• Performing multiplication, or addition is running a simple algorithm. The elementary steps are the operations on digits.

• Finding the greatest common divisor of given two numbers is a computational task. The Euclidean algorithm should be known from BSc.

• Finding the factorization of a given number is a computational task.

Algorithms from school

- Performing multiplication, or addition is running a simple algorithm. The elementary steps are the operations on digits.

- Finding the greatest common divisor of given two numbers is a computational task. The Euclidean algorithm should be known from BSc.

- Finding the factorization of a given number is a computational task. The elementary steps are the basic arithmetical operations.

Refinements

Refinements

- Let $\mathcal{I}$ be the set of possible inputs, and $\mathcal{O}$ be the set of possible outputs.

Refinements

- Let $\mathcal{I}$ be the set of possible inputs, and $\mathcal{O}$ be the set of possible outputs.

Definition: Computational problem

An computational problem is a function $f : \mathcal{I} \to \mathcal{O}$.

Refinements

- Let $\mathcal{I}$ be the set of possible inputs, and $\mathcal{O}$ be the set of possible outputs.

Definition: Computational problem

An computational problem is a function $f : \mathcal{I} \to \mathcal{O}$.

- A computational problem is "just" a valuation of a known function at certain position.

Refinements

• Let $\mathcal{I}$ be the set of possible inputs, and $\mathcal{O}$ be the set of possible outputs.

Definition: Computational problem

An computational problem is a function $f : \mathcal{I} \to \mathcal{O}$.

• A computational problem is "just" a valuation of a known function at certain position.

• We don't say anything on the set of elementary steps. There are several possibilities. The concrete set of basic operations depends on the problem. But be aware this component. When discussing an algorithmic problems you must always know this set.

History

History

- The above description of the notion of algorithm is not a mathematical definition. In the first half of the course we will see many examples for algorithms.

History

• The above description of the notion of algorithm is not a mathematical definition. In the first half of the course we will see many examples for algorithms.

• In the 30's of the XXth century, after long work the mathematical community defined the accepted mathematical notion of algorithm. We will see that in the second half of the semester.

History

- The above description of the notion of algorithm is not a mathematical definition. In the first half of the course we will see many examples for algorithms.

- In the 30's of the XXth century, after long work the mathematical community defined the accepted mathematical notion of algorithm. We will see that in the second half of the semester.

- The present notion is strongly connected to the design of computers.

Example: Computational problems

Example: Computational problems

We can say that we consider the computational problem of FACTORIZATION.

Example: Computational problems

We can say that we consider the computational problem of FACTORIZATION. What do we mean?

Example: Computational problems

We can say that we consider the computational problem of FACTORIZATION. What do we mean?

Example: FACTORIZATION-I

Input: a positive integer.

Example: Computational problems

We can say that we consider the computational problem of FACTORIZATION. What do we mean?

Example: FACTORIZATION-I

Input: a positive integer. Output: its prime divisors with multiplicity.

Example: Computational problems

We can say that we consider the computational problem of FACTORIZATION. What do we mean?

Example: FACTORIZATION-I

Input: a positive integer. Output: its prime divisors with multiplicity.

Example: FACTORIZATION-II

Input: a positive integer.

Example: Computational problems

We can say that we consider the computational problem of FACTORIZATION. What do we mean?

Example: FACTORIZATION-I

Input: a positive integer. Output: its prime divisors with multiplicity.

Example: FACTORIZATION-II

Input: a positive integer. Output: one prime divisor.

Example (cont'd): Computational problems

Example (cont'd): Computational problems

Example: FACTORIZATION-III

Input: a positive integer.

Example (cont'd): Computational problems

Example: FACTORIZATION-III

Input: a positive integer. Output: its smallest prime divisor.

Example (cont'd): Computational problems

Example: FACTORIZATION-III

Input: a positive integer. Output: its smallest prime divisor.

Example: FACTORIZATION-IV

Input: a positive integer and a threshold value t.

Example (cont'd): Computational problems

Example: FACTORIZATION-III

Input: a positive integer. Output: its smallest prime divisor.

Example: FACTORIZATION-IV

Input: a positive integer and a threshold value t. Output: Determine whether our integer has a non-trivial positive divisor at most t.

Example (cont'd): Computational problems

Example: FACTORIZATION-III

Input: a positive integer. Output: its smallest prime divisor.

Example: FACTORIZATION-IV

Input: a positive integer and a threshold value t. Output: Determine whether our integer has a non-trivial positive divisor at most t.

If one knows an algorithm for any of the four problem, then easy to construct a solution to the other three.

Example (cont'd): Computational problems

Example: FACTORIZATION-III

Input: a positive integer. Output: its smallest prime divisor.

Example: FACTORIZATION-IV

Input: a positive integer and a threshold value t. Output: Determine whether our integer has a non-trivial positive divisor at most t.

If one knows an algorithm for any of the four problem, then easy to construct a solution to the other three. Only basic algorithmic theoretical notions are required (for example iteration, binary search).

Coding: The size of input

Coding: The size of input

- In practice we should describe the input for a machine. Also the machine prints out a result, that we need to interpret as an answer/output.

Coding: The size of input

- In practice we should describe the input for a machine. Also the machine prints out a result, that we need to interpret as an answer/output.

- The elements of $\mathcal{I}$ (inputs) and the elements of $\mathcal{O}$ (outputs) should be coded.

Coding: The size of input

- In practice we should describe the input for a machine. Also the machine prints out a result, that we need to interpret as an answer/output.

- The elements of $\mathcal{I}$ (inputs) and the elements of $\mathcal{O}$ (outputs) should be coded. Coding of inputs means $\mathcal{I} \equiv \Sigma^*$.

Coding: The size of input

- In practice we should describe the input for a machine. Also the machine prints out a result, that we need to interpret as an answer/output.

- The elements of $\mathcal{I}$ (inputs) and the elements of $\mathcal{O}$ (outputs) should be coded. Coding of inputs means $\mathcal{I} \equiv \Sigma^*$.

- If the inputs are coded, then the size of the input is an obvious notion:

Coding: The size of input

- In practice we should describe the input for a machine. Also the machine prints out a result, that we need to interpret as an answer/output.

- The elements of $\mathcal{I}$ (inputs) and the elements of $\mathcal{O}$ (outputs) should be coded. Coding of inputs means $\mathcal{I} \equiv \Sigma^*$.

- If the inputs are coded, then the size of the input is an obvious notion: The size is just character counting.

Coding: Example

Coding: Example

Example: SORTING

Given n real numbers $(x_1, \ldots, x_n)$. Determine their ordered sequence.

Coding: Example

Example: SORTING

Given n real numbers $(x_1, \ldots, x_n)$. Determine their ordered sequence.

- The set $\mathbb{R}$ is set of cardinality c/continuum.

Coding: Example

Example: SORTING

Given n real numbers $(x_1, \ldots, x_n)$. Determine their ordered sequence.

- The set $\mathbb{R}$ is set of cardinality c/continuum.
- Σ^* is a set of cardinality $\aleph_0$/countably infinite.

Coding: Example

Example: SORTING

Given n real numbers $(x_1, \ldots, x_n)$. Determine their ordered sequence.

- The set $\mathbb{R}$ is set of cardinality c/continuum.
- Σ^* is a set of cardinality $\aleph_0$/countably infinite.
- "Σ^* is to small to handle all real numbers."

Coding: Example (cont'd)

Coding: Example (cont'd)

- One possibility is that we think about a real number as a result of a measurement.

Coding: Example (cont'd)

- One possibility is that we think about a real number as a result of a measurement. This is always a finite decimal approximation.

Coding: Example (cont'd)

• One possibility is that we think about a real number as a result of a measurement. This is always a finite decimal approximation.

• The size immediately can be interpreted as the number of characters we need to write down the input.

Coding: Example (cont'd)

Coding: Example (cont'd)

- Talk about something different.

Coding: Example (cont'd)

- Talk about something different. One possible set of elementary operation is "comparison of two x_i's".

Coding: Example (cont'd)

- Talk about something different. One possible set of elementary operation is "comparison of two x_i's".
- Now coding can be ignored.

Coding: Example (cont'd)

- Talk about something different. One possible set of elementary operation is "comparison of two x_i's".

- Now coding can be ignored. We do not care how comparison is performed by the machine.

Coding: Example (cont'd)

- Talk about something different. One possible set of elementary operation is "comparison of two x_i's".

- Now coding can be ignored. We do not care how comparison is performed by the machine. We say that the input is considered with exact real arithmetic.

Coding: Example (cont'd)

• Talk about something different. One possible set of elementary operation is "comparison of two x_i's".

• Now coding can be ignored. We do not care how comparison is performed by the machine. We say that the input is considered with exact real arithmetic. **The size of the input is n.**

Coding: Example (cont'd)

• Talk about something different. One possible set of elementary operation is "comparison of two x_i's".

• Now coding can be ignored. We do not care how comparison is performed by the machine. We say that the input is considered with exact real arithmetic. **The size of the input is n.**

• One can say that we work with a virtual machine that has a memory containing boxes.

Coding: Example (cont'd)

• Talk about something different. One possible set of elementary operation is "comparison of two x_i's".

• Now coding can be ignored. We do not care how comparison is performed by the machine. We say that the input is considered with exact real arithmetic. **The size of the input is n.**

• One can say that we work with a virtual machine that has a memory containing boxes. Each box contains a real number and the machine is capable to perform arithmetical operations on the contents.

Coding: Second example

Coding: Second example

The next problem is different.

Coding: Second example

The next problem is different.

Example

Given $x_1, x_2, \ldots, x_n$ k digit numbers. Sort them.

Coding: Second example

The next problem is different.

Example

Given $x_1, x_2, \ldots, x_n$ k digit numbers. Sort them.

The language is explicit.

Coding: Second example

The next problem is different.

Example

Given $x_1, x_2, \ldots, x_n$ k digit numbers. Sort them.

The language is explicit. The input size is $n \cdot k$.

Coding: Second example

The next problem is different.

Example

Given $x_1, x_2, \ldots, x_n$ k digit numbers. Sort them.

The language is explicit. The input size is $n \cdot k$. Elementary operations are digit operations.

Coding: Second example

The next problem is different.

Example

Given $x_1, x_2, \ldots, x_n$ k digit numbers. Sort them.

The language is explicit. The input size is $n \cdot k$. Elementary operations are digit operations. A comparison costs k elementary step.

Coding: Third example

Coding: Third example

Example

Given two integers, calculate their sum / difference / product.

Coding: Third example

Example

Given two integers, calculate their sum / difference / product.

- The elementary steps are the elementary operations between digits.

Coding: Third example

Example

Given two integers, calculate their sum / difference / product.

- The elementary steps are the elementary operations between digits.
- The size of the input character counting.

Coding: Fourth example

Coding: Fourth example

Example

Given two integers. Compute there greatest common divisor.

Coding: Fourth example

Example

Given two integers. Compute there greatest common divisor.

- The elementary steps are basic arithmetical operations on integers.

Coding: Fourth example

Example

Given two integers. Compute there greatest common divisor.

- The elementary steps are basic arithmetical operations on integers.
- For example now taking subtraction is counted as one step.

Coding: Fourth example

Example

Given two integers. Compute there greatest common divisor.

- The elementary steps are basic arithmetical operations on integers.
- For example now taking subtraction is counted as one step.
- The size of the input is character counting.

The relation of view points

The relation of view points

• In the case of computing gcd we said that the natural point of view is that the elementary steps are operations on integers.

The relation of view points

- In the case of computing gcd we said that the natural point of view is that the elementary steps are operations on integers.

- Can we consider digit operation as elementary steps?

The relation of view points

- In the case of computing gcd we said that the natural point of view is that the elementary steps are operations on integers.

- Can we consider digit operation as elementary steps? YES.

The relation of view points

- In the case of computing gcd we said that the natural point of view is that the elementary steps are operations on integers.

- Can we consider digit operation as elementary steps? YES.

- Any high level algorithm can be transformed to digit level algorithm by substituting $x \leftarrow y - z$ step with the basic algorithm of subtraction.

Example: Multiplication of matrices

Example: Multiplication of matrices

Example

Given $A, B \in \mathbb{R}^{n\times n}$, two real matrices of size $n \times n$. Compute $A \cdot B$.

Example: Multiplication of matrices

Example

Given $A, B \in \mathbb{R}^{n\times n}$, two real matrices of size $n \times n$. Compute $A \cdot B$.

- The language suggests "exact real arithmetic".

Example: Multiplication of matrices

Example

Given $A, B \in \mathbb{R}^{n \times n}$, two real matrices of size $n \times n$. Compute $A \cdot B$.

- The language suggests "exact real arithmetic".

- Although our virtual machine doesn't exist, counting multiplications and additions makes sense.

Example: Multiplication of matrices

Example

Given $A, B \in \mathbb{R}^{n\times n}$, two real matrices of size $n \times n$. Compute $A \cdot B$.

- The language suggests "exact real arithmetic".

- Although our virtual machine doesn't exist, counting multiplications and additions makes sense.

- What is the size of the output?

Example: Multiplication of matrices

Example

Given $A, B \in \mathbb{R}^{n\times n}$, two real matrices of size $n \times n$. Compute $A \cdot B$.

- The language suggests "exact real arithmetic".

- Although our virtual machine doesn't exist, counting multiplications and additions makes sense.

- What is the size of the output? Elementary operations are at the level of real numbers. The size of the input is $2n^2$.

Example: Multiplication of matrices

Example

Given $A, B \in \mathbb{R}^{n\times n}$, two real matrices of size $n \times n$. Compute $A \cdot B$.

- The language suggests "exact real arithmetic".

- Although our virtual machine doesn't exist, counting multiplications and additions makes sense.

- What is the size of the output? Elementary operations are at the level of real numbers. The size of the input is $2n^2$. A strange but often useful point of view is to consider n as the size.

Example: Counting triangles

Example: Counting triangles

Example

Given a simple graph G. Determine the number of triangles (cycles of length 3) in G.

Example: Counting triangles

Example

Given a simple graph G. Determine the number of triangles (cycles of length 3) in G.

- How a simple graph is given?

Example: Counting triangles

Example

Given a simple graph G. Determine the number of triangles (cycles of length 3) in G.

- How a simple graph is given? How we can code a simple graph?

Example: Counting triangles

Example

Given a simple graph G. Determine the number of triangles (cycles of length 3) in G.

- How a simple graph is given? How we can code a simple graph?
- There are several possibilities.

Example: Counting triangles

Example

Given a simple graph G. Determine the number of triangles (cycles of length 3) in G.

- How a simple graph is given? How we can code a simple graph?
- There are several possibilities. We highlight two of them.

Coding: Coding graphs I

Coding: Coding graphs I

- The first way to code a simple graph is to describe it adjacency matrix.

Coding: Coding graphs II

Coding: Coding graphs II

- A second, alternative way to code a simple graph is to start with the `list-of-vertices` .

Coding: Coding graphs II

- A second, alternative way to code a simple graph is to start with the `list-of-vertices` .
- Each element of the list there is a `list-of-neighbors`.

Coding: Coding graphs II

• A second, alternative way to code a simple graph is to start with the `list-of-vertices` .

• Each element of the list there is a `list-of-neighbors`.

• In the case of a list, an element of the list contains the "address of the next element" info.

Coding: Coding graphs II

- A second, alternative way to code a simple graph is to start with the `list-of-vertices` .
- Each element of the list there is a `list-of-neighbors`.
- In the case of a list, an element of the list contains the "address of the next element" info. This is a so called pointer field.

Coding: Coding graphs II

• A second, alternative way to code a simple graph is to start with the `list-of-vertices` .

• Each element of the list there is a `list-of-neighbors`.

• In the case of a list, an element of the list contains the "address of the next element" info. This is a so called pointer field. This pointer can have the special value, called "nil", if there is no next element.

Coding: Coding graphs II

• A second, alternative way to code a simple graph is to start with the `list-of-vertices` .

• Each element of the list there is a `list-of-neighbors`.

• In the case of a list, an element of the list contains the "address of the next element" info. This is a so called pointer field. This pointer can have the special value, called "nil", if there is no next element.

• So in the vertex-list every vertex has a reserved memory location, where the corresponding information is stored.

Coding: Coding graphs II

• A second, alternative way to code a simple graph is to start with the `list-of-vertices` .

• Each element of the list there is a `list-of-neighbors`.

• In the case of a list, an element of the list contains the "address of the next element" info. This is a so called pointer field. This pointer can have the special value, called "nil", if there is no next element.

• So in the vertex-list every vertex has a reserved memory location, where the corresponding information is stored. This information is structured.

Coding: Coding graphs II

• A second, alternative way to code a simple graph is to start with the `list-of-vertices` .

• Each element of the list there is a `list-of-neighbors`.

• In the case of a list, an element of the list contains the "address of the next element" info. This is a so called pointer field. This pointer can have the special value, called "nil", if there is no next element.

• So in the vertex-list every vertex has a reserved memory location, where the corresponding information is stored. This information is structured. One can read out the address of the memory location, where the next vertex is stored.

Coding: Coding graphs II

• A second, alternative way to code a simple graph is to start with the `list-of-vertices` .

• Each element of the list there is a `list-of-neighbors`.

• In the case of a list, an element of the list contains the "address of the next element" info. This is a so called pointer field. This pointer can have the special value, called "nil", if there is no next element.

• So in the vertex-list every vertex has a reserved memory location, where the corresponding information is stored. This information is structured. One can read out the address of the memory location, where the next vertex is stored. Also one can read out the address of the memory location, where the first neighbor is stored.

Coding: Coding graphs II

• A second, alternative way to code a simple graph is to start with the `list-of-vertices` .

• Each element of the list there is a `list-of-neighbors`.

• In the case of a list, an element of the list contains the "address of the next element" info. This is a so called pointer field. This pointer can have the special value, called "nil", if there is no next element.

• So in the vertex-list every vertex has a reserved memory location, where the corresponding information is stored. This information is structured. One can read out the address of the memory location, where the next vertex is stored. Also one can read out the address of the memory location, where the first neighbor is stored.

• Any vertex on one of the `list-of-neighbors` corresponds to an edge.

Coding: Coding graphs II

• A second, alternative way to code a simple graph is to start with the `list-of-vertices` .

• Each element of the list there is a `list-of-neighbors`.

• In the case of a list, an element of the list contains the "address of the next element" info. This is a so called pointer field. This pointer can have the special value, called "nil", if there is no next element.

• So in the vertex-list every vertex has a reserved memory location, where the corresponding information is stored. This information is structured. One can read out the address of the memory location, where the next vertex is stored. Also one can read out the address of the memory location, where the first neighbor is stored.

• Any vertex on one of the `list-of-neighbors` corresponds to an edge. In the case of an edge weighted graph the weight of an edge $e = uv$ can be stored at the memory location of v in the `list-of-neighbors` connected to u.

Coding: Coding graphs, size of the input

An interesting remark:

Coding: Coding graphs, size of the input

An interesting remark: Most often, we consider the size of the graph as $|V|+|E|$, or $|V|^2$, or simply $|V|$.

Coding: Final words

Coding: Final words

- Coding is extremely important.

Coding: Final words

- Coding is extremely important.

- In spite of this a high level description of an algorithm does not need to specify it.

Coding: Final words

- Coding is extremely important.

- In spite of this a high level description of an algorithm does not need to specify it.

- Different coding implies different elementary steps.

Coding: Final words

- Coding is extremely important.

- In spite of this a high level description of an algorithm does not need to specify it.

- Different coding implies different elementary steps.

- When we code a simple graph by it adjacency matrix, we can assume that arithmetic operations are elementary steps.

Coding: Final words

- Coding is extremely important.

- In spite of this a high level description of an algorithm does not need to specify it.

- Different coding implies different elementary steps.

- When we code a simple graph by it adjacency matrix, we can assume that arithmetic operations are elementary steps.

- When we code a simple graph by lists, we can assume the going to a memory location described by a pointer is an elementary step.

To summarize

To summarize

- Usually
(1) we will describe a computational problem, (2) we give a high level description of an algorithm, (3) say a few words on coding, (4) describe the elementary steps, although very often the coding determines the most natural operations, that we can consider as basic.

To summarize

- Usually
(1) we will describe a computational problem, (2) we give a high level description of an algorithm, (3) say a few words on coding, (4) describe the elementary steps, although very often the coding determines the most natural operations, that we can consider as basic.

- The high level description of an algorithm $\mathcal{A}$ contains many mathematical/algorithmic ideas. But to consider it as a sequence of elementary steps we need additional ideas. Sometimes this is neglected.

To summarize

- Usually
(1) we will describe a computational problem, (2) we give a high level description of an algorithm, (3) say a few words on coding, (4) describe the elementary steps, although very often the coding determines the most natural operations, that we can consider as basic.

- The high level description of an algorithm $\mathcal{A}$ contains many mathematical/algorithmic ideas. But to consider it as a sequence of elementary steps we need additional ideas. Sometimes this is neglected.

- The reason of this discussion is not to disturb you-

To summarize

- Usually
(1) we will describe a computational problem, (2) we give a high level description of an algorithm, (3) say a few words on coding, (4) describe the elementary steps, although very often the coding determines the most natural operations, that we can consider as basic.

- The high level description of an algorithm $\mathcal{A}$ contains many mathematical/algorithmic ideas. But to consider it as a sequence of elementary steps we need additional ideas. Sometimes this is neglected.

- The reason of this discussion is not to disturb you- The goal is to stimulate you for asking questions, clarify details...

Break

Worst case analysis

Worst case analysis

- Our task: Let $\mathcal{A}$ be an algorithm. Given input ω, run the algorithm and count/estimate the number of elementary steps, you needed for computing the output.

Worst case analysis

- Our task: Let $\mathcal{A}$ be an algorithm. Given input ω, run the algorithm and count/estimate the number of elementary steps, you needed for computing the output.

Notation

$t_{\mathcal{A}}(\omega)$ denotes the exact number of elementary steps, needed to compute the output.

Worst case analysis

• Our task: Let $\mathcal{A}$ be an algorithm. Given input ω, run the algorithm and count/estimate the number of elementary steps, you needed for computing the output.

Notation

$t_{\mathcal{A}}(\omega)$ denotes the exact number of elementary steps, needed to compute the output.

• It should be obvious that $t_{\mathcal{A}}(\omega)$ strongly depends on the size of ω.

Worst case analysis

- Our task: Let $\mathcal{A}$ be an algorithm. Given input ω, run the algorithm and count/estimate the number of elementary steps, you needed for computing the output.

Notation

$t_{\mathcal{A}}(\omega)$ denotes the exact number of elementary steps, needed to compute the output.

- It should be obvious that $t_{\mathcal{A}}(\omega)$ strongly depends on the size of ω.

Notation

$\mathcal{I} = \cup_{s=0}^{\infty} \mathcal{I}_s$, where $\mathcal{I}_s$ denotes the set of inputs of size s.

Time analysis of algorithms

Definition

$$t_{\mathcal{A}}(n) = \max\{t_{\mathcal{A}}(\omega) : \omega \in \mathcal{I}_n\}.$$

Time analysis of algorithms

Definition

$$t_{\mathcal{A}}(n) = \max\{t_{\mathcal{A}}(\omega) : \omega \in \mathcal{I}_n\}.$$

• The definition, above, is very important. If we determine this function, or estimate it then we say that we perform the worst case analysis of our algorithm $\mathcal{A}$.

Time analysis of algorithms

Definition

$$t_{\mathcal{A}}(n) = \max\{t_{\mathcal{A}}(\omega) : \omega \in \mathcal{I}_n\}.$$

• The definition, above, is very important. If we determine this function, or estimate it then we say that we perform the worst case analysis of our algorithm $\mathcal{A}$.

• We consider all the elements of $\mathcal{I}_n$. When $t_{\mathcal{A}}(\omega) < t_{\mathcal{A}}(\omega')$, the input ω' is more complex for $\mathcal{A}$ than ω. When we take the maximum, we consider the worst element of $\mathcal{I}_n$.

Time analysis of algorithms

Definition

$$t_{\mathcal{A}}(n) = \max\{t_{\mathcal{A}}(\omega) : \omega \in \mathcal{I}_n\}.$$

• The definition, above, is very important. If we determine this function, or estimate it then we say that we perform the worst case analysis of our algorithm $\mathcal{A}$.

• We consider all the elements of $\mathcal{I}_n$. When $t_{\mathcal{A}}(\omega) < t_{\mathcal{A}}(\omega')$, the input ω' is more complex for $\mathcal{A}$ than ω. When we take the maximum, we consider the worst element of $\mathcal{I}_n$.

• We can propose any upper bound on $t_{\mathcal{A}}(n)$ as a certificate of $\mathcal{A}$. We guarantee that $\mathcal{A}$ will not execute more elementary steps on any input of size n than our bound.

Example: Counting triangles I

Example: Counting triangles I

Algorithm

Example: Counting triangles I

Algorithm

0) Let $\Delta = 0$.

Example: Counting triangles I

Algorithm

0) Let $\Delta = 0$.

1) Check out each edge, $e = uv$.

Example: Counting triangles I

Algorithm

0) Let $\Delta = 0$.

1) Check out each edge, $e = uv$.

1_e) For all edge $e = uv$ we list the common neighbors of u and v. For each common neighbor, we have found: $\Delta \leftarrow \Delta + 1$.

Example: Counting triangles I

Algorithm

0) Let $\Delta = 0$.

1) Check out each edge, $e = uv$.

1_e) For all edge $e = uv$ we list the common neighbors of u and v. For each common neighbor, we have found: $\Delta \leftarrow \Delta + 1$.

2) Print $\Delta/3$. //We do this, after checking all edges.

Analysis: Counting triangles I

Analysis: Counting triangles I

- We consider the input graph as a data structure describing a graph G by lists.

Analysis: Counting triangles I

- We consider the input graph as a data structure describing a graph G by lists.

- We perform 1_e $|E|$ times.

Analysis: Counting triangles I

- We consider the input graph as a data structure describing a graph G by lists.
- We perform 1_e $|E|$ times.
- 1_e requires D^2 elementary steps.

Analysis: Counting triangles I

- We consider the input graph as a data structure describing a graph G by lists.
- We perform 1_e $|E|$ times.
- 1_e requires D^2 elementary steps.(?)

Analysis: Counting triangles I

- We consider the input graph as a data structure describing a graph G by lists.
- We perform 1_e $|E|$ times.
- 1_e requires D^2 elementary steps.(?)
- A bound on the complete number of elementary steps is

$$D^2 \cdot |E|.$$

Example: Counting triangles II

Example: Counting triangles II

Algorithm

Example: Counting triangles II

Algorithm

1) Compute A_G^3.

Example: Counting triangles II

Algorithm

1) Compute A_G^3.
2) Print $TrA_G^3/6$.

Analysis: Counting triangles II

Analysis: Counting triangles II

- We assume that the input G is given by it adjacency matrix, A_G.

Analysis: Counting triangles II

• We assume that the input G is given by it adjacency matrix, A_G.

• We perform matrix multiplications based on the definition. For one multiplication we $|V|^3$ multiplications and $|V|^3$ additions on two real numbers.

Analysis: Counting triangles II

- We assume that the input G is given by it adjacency matrix, A_G.

- We perform matrix multiplications based on the definition. For one multiplication we $|V|^3$ multiplications and $|V|^3$ additions on two real numbers.

- In order to compute A^3 we perform $4|V|^3$ basic operations on numbers.

Analysis: Counting triangles II

- We assume that the input G is given by it adjacency matrix, A_G.

- We perform matrix multiplications based on the definition. For one multiplication we $|V|^3$ multiplications and $|V|^3$ additions on two real numbers.

- In order to compute A^3 we perform $4|V|^3$ basic operations on numbers.

- A bound on the total number of elementary step we need to do on a graph on vertex set V is

$$4|V|^3 + |V|.$$

Szünet

An important observation

An important observation

- The exact analysis of a complicated algorithm leads to very complex formulas.

An important observation

- The exact analysis of a complicated algorithm leads to very complex formulas.
- What we are interested in is the real time of the running the algorithm of a specific input of known size.

An important observation

- The exact analysis of a complicated algorithm leads to very complex formulas.
- What we are interested in is the real time of the running the algorithm of a specific input of known size.
- How "counting elementary steps" and "real time" are related?

An important observation

• The exact analysis of a complicated algorithm leads to very complex formulas.

• What we are interested in is the real time of the running the algorithm of a specific input of known size.

• How "counting elementary steps" and "real time" are related? There is a hidden constant that depends on the hardware we are using.

An important observation

- The exact analysis of a complicated algorithm leads to very complex formulas.

- What we are interested in is the real time of the running the algorithm of a specific input of known size.

- How "counting elementary steps" and "real time" are related? There is a hidden constant that depends on the hardware we are using.

- If we replace our five years old computer with an up-to-date, new one, than the same algorithm will require less time on the new machine.

An important observation

- The exact analysis of a complicated algorithm leads to very complex formulas.
- What we are interested in is the real time of the running the algorithm of a specific input of known size.
- How "counting elementary steps" and "real time" are related? There is a hidden constant that depends on the hardware we are using.
- If we replace our five years old computer with an up-to-date, new one, than the same algorithm will require less time on the new machine.
- The answer of "theory": Constant factors don't matter, we ignore them.

An important observation

- The exact analysis of a complicated algorithm leads to very complex formulas.

- What we are interested in is the real time of the running the algorithm of a specific input of known size.

- How "counting elementary steps" and "real time" are related? There is a hidden constant that depends on the hardware we are using.

- If we replace our five years old computer with an up-to-date, new one, than the same algorithm will require less time on the new machine.

- The answer of "theory": Constant factors don't matter, we ignore them.

- We need some mathematical notation to reflect this idea.

Big "O" notation

Big "O" notation

Definition

Let $t, f : \mathbb{N} \to \mathbb{R}$. One writes $t = \mathcal{O}(f)$ when for a suitable constant $c > 0$ and threshold value n_0 we have that for any $n > n_0$

$$|t(n)| \leq cf(n)$$

Big "O" notation

Definition

Let $t, f : \mathbb{N} \to \mathbb{R}$. One writes $t = \mathcal{O}(f)$ when for a suitable constant $c > 0$ and threshold value n_0 we have that for any $n > n_0$

$$|t(n)| \leq cf(n)$$

- In this course very often $t(n)$ is the exact worst case analysis of an algorithm, so it is a non-negative function.

Big "O" notation

Definition

Let $t, f : \mathbb{N} \to \mathbb{R}$. One writes $t = \mathcal{O}(f)$ when for a suitable constant $c > 0$ and threshold value n_0 we have that for any $n > n_0$

$$|t(n)| \leq cf(n)$$

- In this course very often $t(n)$ is the exact worst case analysis of an algorithm, so it is a non-negative function.
- $f(n)$ is a „simple" function, like n, n^2, n^{10}, $n \log n$, 2^n, n^n, 2^{n^2}.

Big "O" notation

Definition

Let $t, f : \mathbb{N} \to \mathbb{R}$. One writes $t = \mathcal{O}(f)$ when for a suitable constant $c > 0$ and threshold value n_0 we have that for any $n > n_0$

$$|t(n)| \leq cf(n)$$

- In this course very often $t(n)$ is the exact worst case analysis of an algorithm, so it is a non-negative function.
- $f(n)$ is a „simple" function, like n, n^2, n^{10}, $n \log n$, 2^n, n^n, 2^{n^2}.

Big "O" notation

Definition

Let $t, f : \mathbb{N} \to \mathbb{R}$. One writes $t = \mathcal{O}(f)$ when for a suitable constant $c > 0$ and threshold value n_0 we have that for any $n > n_0$

$$|t(n)| \leq cf(n)$$

- In this course very often $t(n)$ is the exact worst case analysis of an algorithm, so it is a non-negative function.

- $f(n)$ is a „simple" function, like n, n^2, n^{10}, $n \log n$, 2^n, n^n, 2^{n^2}.

- Stop. I wrote $f(n) = n \log n$, and I did not specify the base of the logarithm. What base did I use? Does it matter?

Example

Example

Example

$$18\binom{n}{6} + 127n^4 \log n + \log^{15}(n^{124}) \cdot n + 144 = \mathcal{O}(n^6).$$

Example

Example

$$18\binom{n}{6} + 127n^4 \log n + \log^{15}(n^{124}) \cdot n + 144 = \mathcal{O}(n^6).$$

- $n \log n$ is not defined on $\mathbb{N}$. We do not care about this, since the n_0 threshold in the definition.

Example

Example

$$18\binom{n}{6} + 127n^4 \log n + \log^{15}(n^{124}) \cdot n + 144 = \mathcal{O}(n^6).$$

• $n \log n$ is not defined on $\mathbb{N}$. We do not care about this, since the n_0 threshold in the definition.

• Algorithm theory balances between mathematical correctness and transparency. Algorithm theory/algorithms are used everywhere. Many outsiders are interested in algorithms. So transparency is more important

Example

Example

$$18\binom{n}{6} + 127n^4 \log n + \log^{15}(n^{124}) \cdot n + 144 = \mathcal{O}(n^6).$$

• $n \log n$ is not defined on $\mathbb{N}$. We do not care about this, since the n_0 threshold in the definition.

• Algorithm theory balances between mathematical correctness and transparency. Algorithm theory/algorithms are used everywhere. Many outsiders are interested in algorithms. So transparency is more important

• If you have the feeling that mathematical precision is missing, then stop. Ask questions, clarify the mathematical content, ask for consultation . . .

Big Omega notation

Big Omega notation

- $n^2 = \mathcal{O}(2^n)$ is obviously true.

Big Omega notation

• $n^2 = \mathcal{O}(2^n)$ is obviously true.

• It is true, but at the same time it doesn't give too much information. We need additional notation in order to be able to describe the order of magnitude of a function $t(n)$.

Big Omega notation

- $n^2 = \mathcal{O}(2^n)$ is obviously true.

- It is true, but at the same time it doesn't give too much information. We need additional notation in order to be able to describe the order of magnitude of a function $t(n)$.

Definition

Let $t, f : \mathbb{N} \to \mathbb{R}$. One writes $t = \Omega(f)$ when for a suitable constant $c > 0$ and threshold value n_0 we have that for any $n > n_0$

$$cf(n) \leq t(n).$$

Big Omega notation

- $n^2 = \mathcal{O}(2^n)$ is obviously true.

- It is true, but at the same time it doesn't give too much information. We need additional notation in order to be able to describe the order of magnitude of a function $t(n)$.

Definition

Let $t, f : \mathbb{N} \to \mathbb{R}$. One writes $t = \Omega(f)$ when for a suitable constant $c > 0$ and threshold value n_0 we have that for any $n > n_0$

$$cf(n) \leq t(n).$$

- $f(n)$ must be a function that is positive for a large enough n.

Big Theta notation

Definition

Let $t, f : \mathbb{N} \to \mathbb{R}$. One writes $t = \Theta(f)$ when for suitable constant $c, c' > 0$ and threshold value n_0 we have that for any $n > n_0$

$$cf(n) \leq t(n) \leq c'f(n).$$

Example

Example

Example

$$18\binom{n}{6} + 127n^4 \log n + \log^{15}(n^{124}) \cdot n + 144 = \Theta(n^6).$$

Notion of asymptotics

Notion of asymptotics

When we know the right order of magnitude of $t(n)$, then we might be interested in the hidden constants. Again we introduce a new notion.

Notion of asymptotics

When we know the right order of magnitude of $t(n)$, then we might be interested in the hidden constants. Again we introduce a new notion.

Definition

Let $t, f : \mathbb{N} \to \mathbb{R}$. One writes $t(n) \sim c \cdot f(n)$

$$\lim_{n\to\infty} t(n)/f(n) = c.$$

Notion of asymptotics

When we know the right order of magnitude of $t(n)$, then we might be interested in the hidden constants. Again we introduce a new notion.

Definition

Let $t, f : \mathbb{N} \to \mathbb{R}$. One writes $t(n) \sim c \cdot f(n)$

$$\lim_{n\to\infty} t(n)/f(n) = c.$$

In other words the meaning of $t(n) \sim c \cdot f(n)$ is that

$$t(n) = c \cdot f(n) + o(f(n)).$$

Little "o" notion

Little "o" notion

Definition

Let $t, f : \mathbb{N} \to \mathbb{R}$. One writes $t(n) = o(f(n))$

$$\lim_{n\to\infty} t(n)/f(n) = 0.$$

Little "o" notion

Definition

Let $t, f : \mathbb{N} \to \mathbb{R}$. One writes $t(n) = o(f(n))$

$$\lim_{n\to\infty} t(n)/f(n) = 0.$$

For large enough n the term $o(f(n))$ in $f(n) + o(f(n))$ is an error/remainder term next to $f(n)$, the main term.

Example

Example

$$18\binom{n}{6} + 127n^4 \log n + \log^{15}(n^{124}) \cdot n + 144 \sim \frac{18}{6!}n^6,$$

or we can write

$$18\binom{n}{6} + 127n^4 \log n + \log^{15}(n^{124}) \cdot n + 144 = \frac{18}{6!}n^6 + \mathcal{O}(n^5),$$

to express more information.

This is the end!

Thank you for your attention!