Algorithms and Complexity

1. Basics

Marc Hellmuth

University of Stockholm

Central Questions:

- What is an "algorithm" and what a "datastructure"?
- What does "complexity" mean and what "efficiency"?
  [efficiency heavily depends on complexity!]

Central Questions:

- What is an “algorithm” and what a “datastructure”?
- What does “complexity” mean and what “efficiency”?
  [efficiency heavily depends on complexity!]

algorithm (informal):

The word ’algorithm’ has its roots in the name of Persian mathematician Muhammad ibn Musa al-Khwarizmi.

He wrote a fundamental treatise on the “Hindu–Arabic numeral system” which was translated into Latin during the 12th century.
Here: al-Khwarizmi was translated into Algorizmi

(780–850) Persian mathematician, astronomer, geographer, . . .

Central Questions:

- What is an “algorithm” and what a “datastructure”?
- What does “complexity” mean and what “efficiency”?
  [efficiency heavily depends on complexity!]

algorithm (informal):

The first computer program was written by Ada Lovelace for the “Analytical Engine” [design for a simpler mechanical computer] by Charles Babbage to compute Bernoulli numbers.

(1815-1852) English mathematician

Central Questions:

- What is an “algorithm” and what a “datastructure”?
- What does “complexity” mean and what “efficiency”?
  [efficiency heavily depends on complexity!]

algorithm (informal):

An algorithm is every well-defined computable (?) procedure which

- has as input a (finite) set of values
- applies a sequence of operations on values
- has as output a (finite) set of values

Central Questions:

- What is an “algorithm” and what a “datastructure”?
- What does “complexity” mean and what “efficiency”?
  [efficiency heavily depends on complexity!]

algorithm (informal):

An algorithm is every well-defined computable (?) procedure which

- has as input a (finite) set of values
- applies a sequence of operations on values
- has as output a (finite) set of values

Examples from “real world”: Addition, Translation of Roman numerals into decimal numbers, cooking recipe, turn-by-turn directions (nav), . . .)

Central Questions:

- What is an “algorithm” and what a “datastructure”?
- What does “complexity” mean and what “efficiency”?
  [efficiency heavily depends on complexity!]

algorithm (informal):

An algorithm is every well-defined computable (?) procedure which

- has as input a (finite) set of values
- applies a sequence of operations on values
- has as output a (finite) set of values

Examples from “real world”: Addition, Translation of Roman numerals into decimal numbers, cooking recipe, turn-by-turn directions (nav), . . .)

We will make this later more precise: Turing machines

Central Questions:

- What is an “algorithm” and what a “datastructure”?
- What does “complexity” mean and what “efficiency”?
  [efficiency heavily depends on complexity!]

datastructure. . .

. . . is object to store & organize data to get efficient access/modification/organization of data

efficiency = speed in terms of runtime and economical use of resources (space)

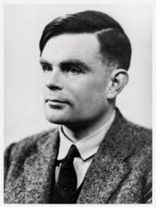
Central Questions:

- What is an “algorithm” and what a “datastructure”?
- What does “complexity” mean and what “efficiency”?
  [efficiency heavily depends on complexity!]

complexity and **efficiency** later, before we need to understand the term “algorithm”

What is an algorithm?

The formal definition of algorithm goes back to Alan Turing



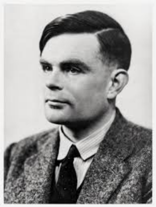
(1912-1954) English mathematician, computer scientist, logician, . . .

Excellent overview: https://plato.stanford.edu/entries/turing-machine/

TM simulator: http://turingmachine.io/

What is an algorithm?

The formal definition of algorithm goes back to Alan Turing



(1912-1954) English mathematician, computer scientist, logician, . . .

Definition 1 (algorithm)

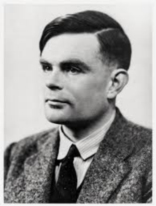
A calculation rule for a problem is called **algorithm** if and only if there is a Turing machine equivalent to this calculation rule which stops for every input that has solution

Excellent overview: https://plato.stanford.edu/entries/turing-machine/

TM simulator: http://turingmachine.io/

What is an algorithm?

The formal definition of algorithm goes back to Alan Turing



(1912-1954) English mathematician, computer scientist, logician, . . .

Definition 1 (algorithm)

A calculation rule for a problem is called **algorithm** if and only if there is a Turing machine equivalent to this calculation rule which stops for every input that has solution

What is a Turing machine ???

Essentially, a Turing machine is theoretical machine that simulates the operating principles of a computer (central processing unit = CPU)

Excellent overview: https://plato.stanford.edu/entries/turing-machine/

TM simulator: http://turingmachine.io/

Turing Machine

Definition 2 (Turing machine (TM))

A TM is a 5-tupel $M = (Q, \Sigma, q_0, \delta, F)$ with

- Q: finite set of states
- Σ: finite set of symbols
- q_0: initial state
- δ: transition function $\delta: Q \times \Sigma \to Q \times \Sigma \times \{L, R\}$

  [If $\delta(q, s)$ not defined then HALT]
- $F \subseteq Q$: set of accepting states

In addition there is a special blank symbol "␣" which is not allowed to be part of the input string at the initial state. The blank symbol is the only symbol to be allowed to occur "infinitely often" after the final state is reached.
L=left, R=right

example $\delta : (q_i, x) \mapsto (q_j, y, l)$ means, if at start of state q_i the head reads symbol x then replace x by y, move head by one position left (L) and go to state q_j.

Turing machine

WHITEBOARD: Idea TM + finite state representation

Turing machine

WHITEBOARD: Idea TM + finite state representation

Definition 3

Σ^* denotes the set of all words/strings over the alphabet Σ

$L \subseteq \Sigma^*$ is called language (over Σ)

For a TM M, let $\mathsf{L}(M) = \{w \mid M \text{ accepts } w\}$.

A language $L \subseteq \Sigma^*$ is TM-recognizable [often: recursively enumerable language] $\iff$ there is TM M such that $L = \mathsf{L}(M)$.

Note, if $w \in \mathsf{L}(M) \setminus \Sigma^*$, then M with input w may never halt or rejects&halt

Turing machine

WHITEBOARD: Idea TM + finite state representation

Definition 3

Σ^* denotes the set of all words/strings over the alphabet Σ

$L \subseteq \Sigma^*$ is called language (over Σ)

For a TM M, let $\mathsf{L}(M) = \{w \mid M \text{ accepts } w\}$.

A language $L \subseteq \Sigma^*$ is TM-recognizable [often: recursively enumerable language] $\iff$ there is TM M such that $L = \mathsf{L}(M)$.

Note, if $w \in \mathsf{L}(M) \setminus \Sigma^*$, then M with input w may never halt or rejects&halt
WHITEBOARD TM-Exmpl 1, 2

Further examples are in the script (Exercise/Tutorial)

Turing machine

WHITEBOARD: Idea TM + finite state representation

Definition 3

Σ^* denotes the set of all words/strings over the alphabet Σ

$L \subseteq \Sigma^*$ is called language (over Σ)

For a TM M, let $\mathsf{L}(M) = \{w \mid M \text{ accepts } w\}$.

A language $L \subseteq \Sigma^*$ is TM-recognizable [often: recursively enumerable language] $\iff$ there is TM M such that $L = \mathsf{L}(M)$.

Note, if $w \in \mathsf{L}(M) \setminus \Sigma^*$, then M with input w may never halt or rejects&halt
WHITEBOARD TM-Exmpl 1, 2

Further examples are in the script (Exercise/Tutorial)

Turing essentially showed that any computation that can be done by mechanical means (and thus, a computer) can be formulated by a TM, i.e., anything a real computer can compute can be formulated as TM.

Note: any computer stores input, programs, etc as a finite binary string and every (decision) problem can be expressed as a language over $\Sigma = \{0, 1\}$

Turing machine

Definition 4

Let $\Sigma = \{0, 1\}$. A decision problem is a language $L \subseteq \Sigma^*$ that contains all binary words that encode an instance of the problem where the answer is “yes”.

Dec. Probl. is decidable if there is a TM that halts for every input $x \in \Sigma^*$ and satisfies: TM accepts x if and only if $x \in L$. In this case, only ”accept=YES” and ”reject=NO” the possible final states since TM halts.

Example: Is string of form $O^N 1^N$? Does binary string represent even number?

Turing machine

Definition 4

Let $\Sigma = \{0,1\}$. A decision problem is a language $L \subseteq \Sigma^*$ that contains all binary words that encode an instance of the problem where the answer is "yes".

Dec. Probl. is decidable if there is a TM that halts for every input $x \in \Sigma^*$ and satisfies: TM accepts x if and only if $x \in L$. In this case, only "accept=YES" and "reject=NO" the possible final states since TM halts.

Example: Is string of form $O^N 1^N$? Does binary string represent even number?

Often we think of Turing machines not only as acceptors of languages but as as computers of functions. The input to the function is the initial content of the tape and the output is the final content of the tape when the TM reaches an accepting state.

Definition 5

Let Σ_1, Σ_2 be alphabets. A function $f\colon \Sigma_1^* \to \Sigma_2^*$ is computable, if there is a TM s.t. $\forall x \in \Sigma_1^*$ this TM halts and has a output $f(x)$.

Example: If string s is of the form $O^N 1^N$ then $f(s) = x^N y^N$

Turing machine

There are several variations of TM:

- one tape vs many tapes
- infinite on both “end”
- allow head to stay on same place
- . . .

Question: Can this lead to more powerful TMs?

Answer: No, all these variants can be shown to be equivalent in the sense that they can be simulated by a “usual” TM (= universal TM) [without proof]

Turing machines and algorithm

The latter should give you a general idea of the terms “algorithm” and “computability”

Of course, to check whether we have indeed an “algorithms” it is quite a bit work to rephrase this in terms of languages and TMs.

Turing machines and algorithm

The latter should give you a general idea of the terms “algorithm” and “computability”

Of course, to check whether we have indeed an “algorithms” it is quite a bit work to rephrase this in terms of languages and TMs.

However, we can give a simplified summary:

For use it is enough to call a procedure an algorithm if it satisfies:

1. procedure is described in unique way by a finite text and every step is executible [transition function]
2. input is finite set of values [initial state q_0]
3. In every step only a finite amount of memory is used
4. procedure terminates and has some finite set of values as output [halts]

Central Questions . . .

. . . we want to address in this course

- Can all problems be solved by algorithms?
- Which problems can be solved efficiently (polynomial-time) and which not?
- Does the “structure” of problems imply ways to efficiently solve them or to design efficient datastructure?
- What if we can only approximate solutions and how can we prove that finding exact solutions is not feasible?

Before we can start to answer all these questions, we introduce graphs (=mathematical structures that we will heavily use in this course)

Graphs - WHITEBOARD / see DA4006-script

- directed/undirected graph $G = (V, E)$
- adjacent and incident
- neighborhood and degree
- (induced)subgraph $H \subseteq G$
- complete graph K_n
- complement $\overline{G}$ of G
- isomorphic graphs $G_1 \simeq G_2$ and isomorphism
- (simple) path and cycles
- (strongly) connected component
- DAG / forest / tree

Exercise:

- $\sum_{v \in V} \deg(v) = 2|E|$ for undirected graph $G = (V, E)$
- Every undirected graph has an even number of vertices of odd degree
- Every undirected graph $G = (V, E)$ with $|V| > 1$ contains two vertices of the same degree

Graphs - WHITEBOARD / see DA4006-script

Theorem 1.1

For an undirected graph $G = (V, E)$ the following statements are equivalent

1. *G is a tree*
2. *For all $u, v \in V$ exists a unique simple u-v-path in G*
3. *G is connected but $G - e := (V, E \setminus \{e\})$ is disconnected $\forall e \in E$*
4. *G is acyclic, but $G + e := (V, E \cup \{e\})$ is contains a cycle $\forall e \notin E$*

Theorem 1.2

Let $G = (V, E)$ be a connected undirected graph. Then, $|G|$ is a tree if and only if $|E| = |V| - 1$.

Corollary 1.3

Every connected undirected graph $G = (V, E)$ has a tree $T = (V, F)$ as subgraph (=spanning tree) and $|E| \geq |V| - 1$.

Proofs on whiteboard or exercise / see DA4006-script

Graph representation (visual, Adjacency matrix, Incidence matrix, . . .)