Computational complexity

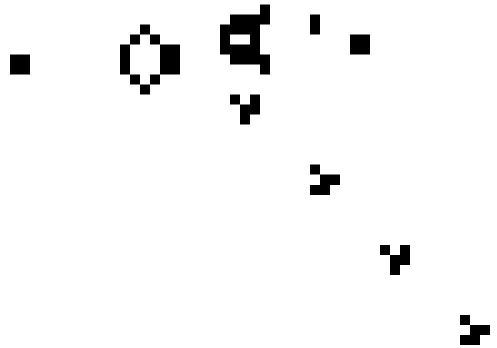
Petr Kurapov

Fall 2024

MIPT

Computational complexity

- Notion of computation: numbers juggling while following some rules
- 20th century – precise definition
- Diverse physical and mathematical systems: Turing machines, lambda calculus, cellular automata, pointer machines, the game of life, …
- *Standard* Universal electronic computer capable of executing *any* program



Source: https://en.wikipedia.org/wiki/Cellular_automaton (Gosper's Glider Gun creating "gliders" in the cellular automaton Conway's Game of Life)

Computational complexity

- Some of the problems appeared to be *uncomputable*
- *Computational efficiency* – how to quantify?

Computational complexity

- Main objective:
  - Estimate resource (time, memory, communication, randomness, etc.) amount required to solve a problem – computational efficiency.
- Questions examples:
  - Relations between computation problems
  - Worst-case vs average-case
  - Approximation benefit

Computational efficiency: a simple example

- a*b
  - Add to a b – 1 times.
  - Grade-school algorithm
- 422 vs 3 + 2

			5	7	7
			4	2	3
		1	7	3	1
	1	1	5	4	
2	3	0	8		
2	4	4	0	7	1

Computational efficiency: a simple example

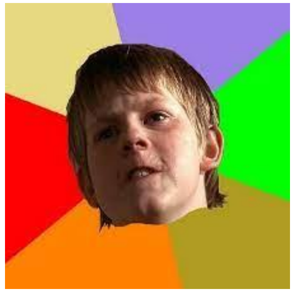
- a*b
  - Add to a b – 1 times.
  - Grade-school algorithm
- 422 vs 3 + 2

			5	7	7
			4	2	3
		1	7	3	1
	1	1	5	4	
2	3	0	8		
2	4	4	0	7	1

Quantify efficiency as how number of basic ops scales with inputs

- Ie. $2n^2\ vs\ n10^{n-1}, \mathrm{n}\ - \mathrm{num\ of\ digits}$

VS



Matrix multiplication: Ideal case & real-world algorithms

- Lower bound: $\Omega(n^{\omega+o(1)})$
- $2 \leq \omega \leq 3$

Even simple problems may have non-obvious algorithms that were not discovered for centuries

*details will be covered later in the course when we have all the required tools to properly analyze

- 1969: $n^{2.807}$ (Strassen)
- 1978 $n^{2.796}$ (Pan)
- 1979 $n^{2.780}$ (Bini, Capovani, Romani)
- 1981 $n^{2.522}$ (Schönhage)
- 1981 $n^{2.517}$ (Romani)
- 1981: $n^{2.496}$(Coppersmith, Winograd)
- 2010: $n^{2.37293}$(Stothers)
- 2012: $n^{2.372873}$ (todo)
- 2014: $n^{2.3728639}$ (Le Gall)
- 2020: $n^{2.3728596}$ (Williams)
- 2022: $n^{2.37188}$ (Duan, Wu and Zhou)

Computational efficiency

- Dinner party:
  - Find the largest subset of guests given a list of pairs who don’t get along with each other so that every pair of invitees have a good relationship

- Obvious yet inefficient solution: check all 2^n subsets – *exhaustive search*. (get yourself a datacenter if you want a 70+ person party!)
  - Any better algorithm?

More questions than answers…

- Can we prove the algorithm we came up with is the best possible?
- Can we replace an exhaustive search with a better alternative?
- Can an algorithm use randomness to speed up computation?
- Can hard problems become easier if we allow errors or approximations on small input subsets?
- Can we use hard problems for constructing cryptographic protocols that are unbreakable?
- Can we use quantum mechanics to build faster computers?
- Can we generate mathematical proofs automatically?

Types of complexity

- Decompressor: $D(x) = y : x, y \in \{0,1\}, \text{so D: } \{0,1\} \to \{0,1\}$
- Kolmogorov complexity of an object is the length of a shortest computer program that produces the object as output:
$$KS_D(x) = \min\{len(y) | D(y) = x\}$$
- *Berry paradox*: наименьшее число, которое нельзя определить фразой из не более, чем тринадцати русских слов
- Shannon's entropy: $H(x) = \sum_{x \in \chi} p_x \log 1/p_x , P(X = x) = p_x$
- Occam's razor or Minimal Description Length (MDL)

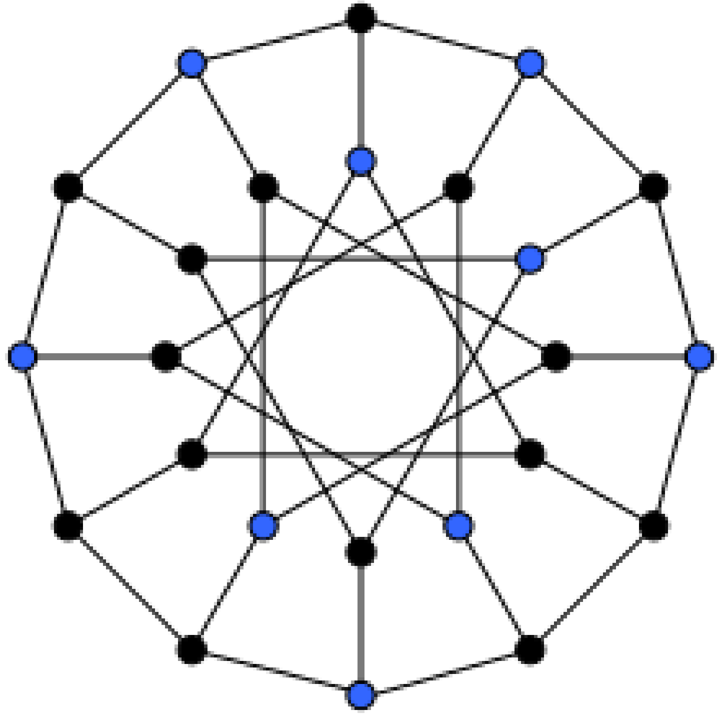
Shannon Information and Kolmogorov Complexity: https://arxiv.org/abs/cs/0410002

Notations & representation

- String – finite ordered tuple of elements from alphabet S.
- S^n- set of n-length strings over S
- $S^* = \bigcup_{n\geq 0} S^n$
- Binary strings: {0, 1}*
- Functions with string inputs and outputs
- Any input can be encoded with binary strings
- Concat of strings a,b can be encoded as a#b, and then $1 \rightarrow 11, 0 \rightarrow 01, \# \rightarrow 00$
- $L_f = \{x: f(x) = 1\}\ of\ \{0,1\}^*$- decision problem/language (example?)

Notations & representation

- $L_f = \{x: f(x) = 1\}\ of\ \{0,1\}^*$- decision problem/language
- $ISet = \{(G,k): \exists S \subseteq V_G: |S| \geq k\ \&\ \forall u,v \in S, \overline{uv} \notin E_G\}$ (independent set – a set of vertices no two of which are adjacent).



Big-Oh quick recap

- $f, g: \mathbb{N} \rightarrow \mathbb{N}$
  - $f = O(g)\ if\ there's\ c: f(n) \leq cg(n)$
  - $f = \Omega(g)\ if\ g = O(f)$
  - $f = \Theta(g)\ if\ both\ f = O(g)\ \&\ f = \Omega(g)$
  - $f = o(g)\ if\ \forall \varepsilon > 0\ \exists n : f(n) \leq \varepsilon g(n)$
  - $f = \omega(g)\ if\ g = o(f)$

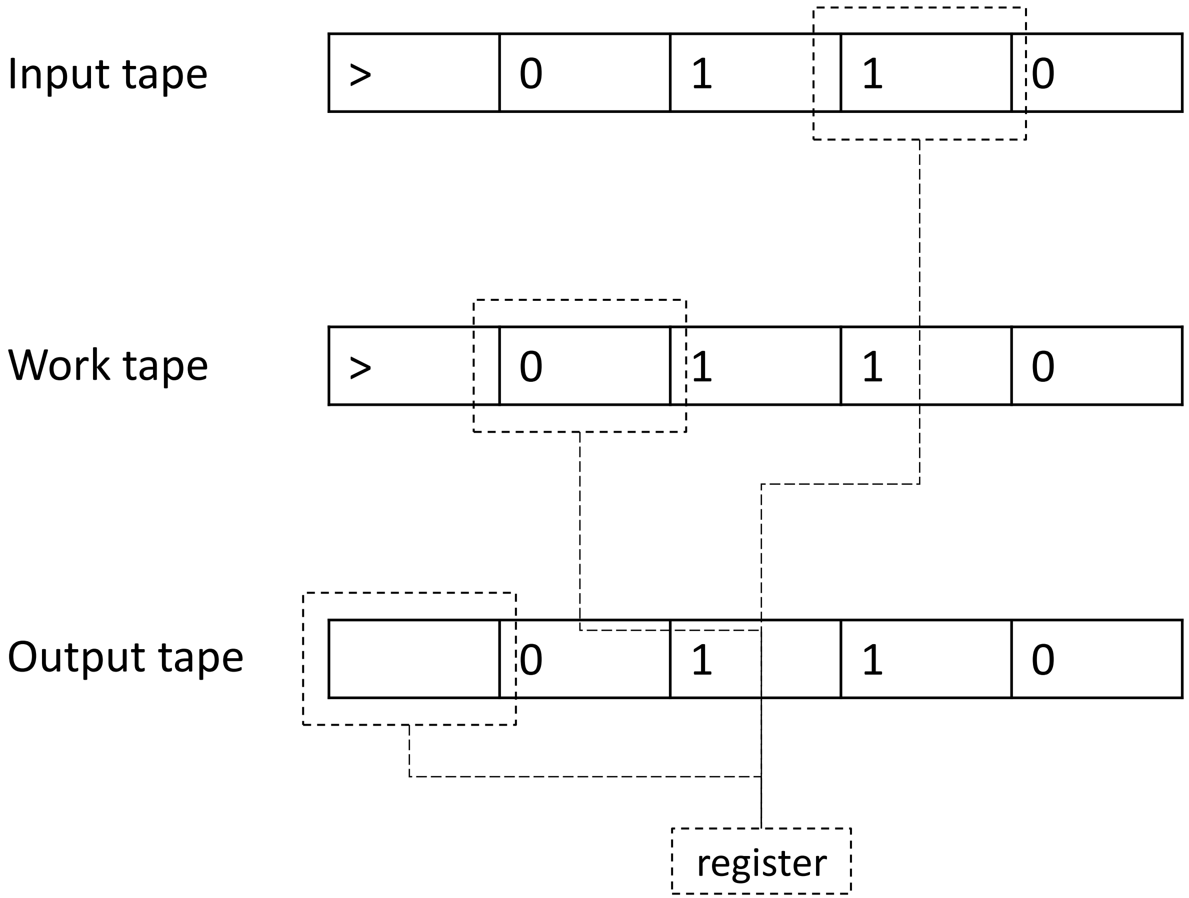
- Example: $f(n) = f(n-1) + 5$

Computation model

- Computation – numbers manipulation under some rules using a *scratch pad* with intermediate results
- Mathematical model?
- Turing machine

Turing machine

- Have a function $f: \{0,1\}^* \rightarrow \{0,1\}$. An **algorithm** – set of *fixed* rules:
  - Read input bit
  - Read a bit/symbol from scratch pad (allows greater alphabet)
  - Write to scratch pad
  - Stop and output 0/1 or next rule
- Running time – number of basic ops, asymptotic $T(n)$
- Universal Turing machine U, works in $O(T(|x|)\log(T(|x|)))$



Turing machine

- Scratch pad = k-tape (1 input, work, 1 output), each has its own *head*, stores alphabet $\Gamma = \{\rhd, \square, 0,1\}$
- Register for states (Q, finite)
- Transition function δ

IF			**THEN**			
Input symbol read	Work/out tape symbol read	Current state	Move input head	New work/out tape symbol	Move work/out tape	New state
A	B	Q	Right	B’	Left	Q’

Turing machine

- M – Turing machine, T: $\mathbb{N} \rightarrow \mathbb{N}$
- M computes *f* if:
  - For every {0, 1}* and M initialized to start config M *halts*
  - *f(x)* written to output tape
- M computes *f* in *T(n)* time if computation on every *x* requires *T(|x|)* at most
- PAL example
- Time constructible (n, nlogn, n^2):
  - $T(n) \geq n$
  - $\exists M, computes\ x \rightarrow T(x)_{binary}\ in\ T(n)$

Turing machine: important notes

- The set of rules is fixed and remains the same for all inputs
- Execution time – asymptotic number of steps required for an algorithm to reach a “finished” state
- Any machine can have a string *description* (for invalid machines we can set an empty MT). This means that a machine can be an input to another machine.

$$Machine \ \approx algorithm$$

Turing machine: robustness

- Turing machines (TM) do not change efficiency properties on its structure changes
- By alphabet: assume TM M with alphabet Γ, a computable f is computable in $4\log|\Gamma|\,T(n)$ using $\Gamma = \{\rhd, \square, 0,1\}$
- Single tape can simulate k-tape in $5kT(n)^2$
- How to simulate a bidirectional work tape machine?

Turing machine: robustness

- Turing machines (TM) do not change efficiency properties on its structure changes
- By alphabet: assume TM M with alphabet Γ, a computable f is computable in $4\log|\Gamma|\, T(n)$ using $\Gamma = \{\rhd, \square, 0, 1\}$
- Single tape can simulate k-tape in $5kT(n)^2$
- Bidirectional tape can be simulated with unidirectional with Γ^2 alphabet in $4T(n)$
  - What would be the slowdown?

Universal Turing machine

- $\exists$TM U: $\forall x, a \in \{0,1\}^*$ $U(x,a) = M_a(x)$, halts in $CTlogT$
- Machine description as input
- 3-tape, simplest alphabet transformation

Input tape

>	0	1	1	0	1	0	1		

M work tape simulation	

M description	

Current state of M	

Output tape

>	0	1	1	0	1	0	1		

Undecidable functions

- UC: {0,1}* -> {0, 1}:
  - $UC\ \ ::\ \forall a \in \{0,1\}^*\ if\, M_a(a) = 1\ then\ UC(a) = 0, and\ 1\ otherwise$
- Diagonalization

MT description {0,1}*

Input parameter

	0	1	a
0	$M_a(a) \to 1 - M_a(a)$	*	$M_0(a)$
1	*	…	*
a	*	*	…

Halting problem

- HALT: $\langle a, x\rangle, 1\ if\ M_a\ halts\ on\ x, 0\ otherwise$
- HALT is not computable by any TM
- Reduction:
  - $M_{UC}\ on\ input\ a\ runs\ M_{HALT(a,a)}$
    - If result is 0 (M_a doesn't halt on a), then M_{UC} outputs 1
    - Otherwise M_{UC} runs universal TM to compute $b = M_a(\mathrm{a})$, if b = 1 M_{UC} outputs 0, and vise versa
  - As $M_{HALT(a,a)}$ outputs $HALT(a, a)$ in finite number of steps $M_{UC}(a)$ outputs $UC(a)$
- UC is reducible to HALT + Gödel's theorem

Class P

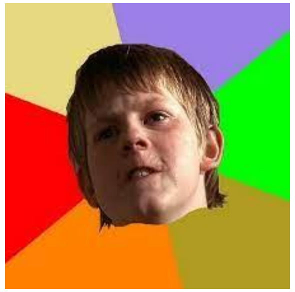
- Complexity class – set of functions that can be computed within resource bounds
- Machine decides language $L \subseteq \{0,1\}^*$ if computes $f_L: \{0,1\}^* \rightarrow \{0,1\}$;
  - $f_L(x) = 1 \leftrightarrow x \in L$
- DTIME: T: $\mathbb{N} \rightarrow \mathbb{N}$, L is in DTIME(T(n)) if $\exists M$ that decides L and runs in $cT(n)$
- $P = \bigcup_{c \geq 1} DTIME(n^c)$ -- note: DTIME(n^{100}) is also in P. In practice if a problem is in P there is an algorithm with $\sim n^5$ complexity
- $ISet \in P$?

Other model?

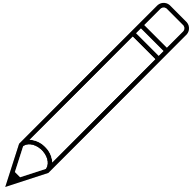
- **Church-Turing (CT) thesis**: every physically realizable device can be simulated with a TM

Universal Turing machine

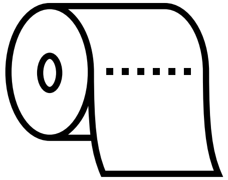
*lives forever



+

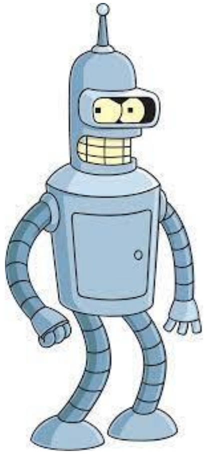


+



*infinite

Cannot outperform



Few notes

- Worst-case exact computation is too strict for practical purposes (usually addressed by a sort of mean complexity)
- Physics:
  - Precision (int vs float)
  - Randomness (BPP – P analogue)
  - Quantum mechanics (BQP)
- Decision problems are too limited

Resources

- Computational Complexity: A Modern Approach (https://theory.cs.princeton.edu/complexity/book.pdf)
- Колмогоровская сложность и алгоритмическая случайность (https://www.mccme.ru/free-books/shen/kolmbook.pdf)

Backup