Algorithmic Information Theory and Kolmogorov Complexity

Alexander Shen, *
Uppsala University, Computing Science Department,
Independent University of Moscow, shen@mccme.ru

November 19, 2007

Abstract

This document contains lecture notes of an introductory course on Kolmogorov complexity. They cover basic notions of algorithmic information theory: Kolmogorov complexity (plain, conditional, prefix), notion of randomness (Martin-Löf randomness, Mises–Church randomness), Solomonoff universal a priori probability and their properties (symmetry of information, connection between a priori probability and prefix complexity, criterion of randomness in terms of complexity) and applications (incompressibility method in computational complexity theory, incompleteness theorems).

Contents

1 Compressing information **3**

2 Kolmogorov complexity **3**

3 Optimal decompression algorithm **3**

4 The construction of optimal decompression algorithm **4**

5 Basic properties of Kolmogorov complexity **4**

6 Algorithmic properties of K **5**

7 Complexity and incompleteness **5**

8 Algorithmic properties of K (continued) **6**

9 An encodings-free definition of complexity **6**

10 Axioms of complexity **7**

11 Complexity of pairs **8**

12 Conditional complexity **8**

13 Pair complexity and conditional complexity **9**

14 Applications of conditional complexity **11**

*Supported by STINT, The Swedish Foundation for International Cooperation in Research and Higher Education, grant Dnr 99/621

15 Incompressible strings **11**

16 Computability and complexity of initial segments **11**

17 Incompressibility and lower bounds **13**

18 Incompressibility and prime numbers **14**

19 Incompressible matrices **14**

20 Incompressible graphs **15**

21 Incompressible tournaments **15**

22 Discussion **15**

23 k- and $k+1$-head automata **16**

24 Heap sort: time analysis **17**

25 Infinite random sequences **18**

26 Classical probability theory **18**

27 Strong Law of Large Numbers **19**

28 Effectively null sets **19**

29 Maximal effectively null set **20**

30 Gambling and selection rules **21**

31 Selection rules and Martin-Löf randomness **21**

32 Probabilistic machines **23**

33 A priori probability **25**

34 Prefix decompression **26**

35 Prefix complexity and length **26**

36 A priory probability and prefix complexity **27**

37 Prefix complexity of a pair **28**

38 Strong law of large numbers revisited **30**

1 Compressing information

Everybody now is familiar with compressing/decompressing programs such as `zip`, `gzip`, `compress`, `arj`, etc. A compressing program can be applied to any file and produces the "compressed version" of that file. If we are lucky, the compressed version is much shorter than the original one. However, no information is lost: the decompression program can be applied to the compressed version to get the original file.

[Question: A software company advertises a compressing program and claims that this program can compress *any* sufficiently long file to at most 90% of its original size. Would you buy this program?]

How compression works? A compression program tries to find some regularities in a file which allow to give a description of the file which is shorter than the file itself; the decompression program reconstructs the file using this description.

2 Kolmogorov complexity

The Kolmogorov complexity may be roughly described as "the compressed size". However, there are some differences. The technical difference is that instead of files (which are usually byte sequences) we consider bit strings (sequences of zeros and ones). The principal difference is that in the framework of Kolmogorov complexity we have no *compression* algorithm and deal only with the *decompression* algorithm.

Here is the definition. Let U be any algorithm whose inputs and outputs are binary strings. Using U as a decompression algorithm, we define the complexity $K_U(x)$ of a binary string x with respect to U as follows:

$$K_U(x) = \min\{|y| \,|\, U(y) = x\}$$

(here $|y|$ denotes the length of a binary string y). In other words, the complexity of x is defined as the length of the shortest description of x if each binary string y is considered as a description of $U(y)$

Let us stress that $U(y)$ may be defined not for all y's and that there are no restrictions on time necessary to compute $U(y)$. Let us mention also that for some U and x the set in the definition of K_U may be empty; we assume that $\min(\emptyset) = +\infty$.

3 Optimal decompression algorithm

The definition of K_U depends on U. For the trivial decompression algorithm $U(y) = y$ we have $K_U(x) = |x|$. One can try to find better decompression algorithms, where "better" means "giving smaller complexities". However, the number of short descriptions is limited: There is less than 2^n strings of length less than n. Therefore, for any fixed decompression algorithm the number of words whose complexity is less than n does not exceed $2^n - 1$. One may conclude that there is no "optimal" decompression algorithm because we can assign short descriptions to some string only taking them away from other strings. However, Kolmogorov made a simple but crucial observation: there is *asymptotically optimal* decompression algorithm.

Definition 1 *An algorithm U is asymptotically not worse than an algorithm V if $K_U(x) \leqslant K_V(x) + C$ for come constant C and for all x.*

Theorem 1 *There exists an decompression algorithm U which is asymptotically not worse than any other algorithm V.*

Such an algorithm is called *asymptotically optimal* one. The complexity K_U with respect to an asymptotically optimal U is called *Kolmogorov complexity*. The Kolmogorov complexity of a string x is

denoted by $K(x)$. (We assume that some asymptotically optimal decompression algorithm is fixed.) Of course, Kolmogorov complexity is defined only up to $O(1)$ additive term.

The complexity $K(x)$ can be interpreted as the amount of information in x or the "compressed size" of x.

4 The construction of optimal decompression algorithm

The idea of the construction is used in the so-called "self-extracting archives". Assume that we want to send a compressed version of some file to our friend, but we are not sure he has the decompression program. What to do? Of course, we can send the program together with the compressed file. Or we can append the compressed file to the end of the program and get an executable file which will be applied to its own contents during the execution).

The same simple trick is used to construct an universal decompression algorithm U. Having an input string x, the algorithm U starts scanning x from left to right until it founds some program p written in a fixed programming language (say, Pascal) where programs are self-delimiting (so the end of the program can be determined uniquely). Then the rest of x is used as an input for p, and $U(x)$ is defined as the output of p.

Why U is (asymptotically) optimal? Consider any other decompression algorithm V. Let v be a Pascal program which implements V. Then

$$K_U(x) \leqslant K_V(x) + |v|$$

for any string x. Indeed, if y is V-compressed version of x (i.e., $V(y) = x$), then vy is U-compressed version of x (i.e., $U(vy) = x$) which is only $|v|$ bits longer.

5 Basic properties of Kolmogorov complexity

(a) $K(x) \leqslant |x| + O(1)$

(b) The number of x's such that $K(x) \leqslant n$ is equal to 2^n up to a bounded factor separated from zero.

(c) For any computable function f there exists a constant c such that

$$K(f(x)) \leqslant K(x) + c$$

(for any x such that $f(x)$ is defined).

(d) Assume that for any natural n a finite set V_n containing not more than 2^n elements is given. Assume that the relation $x \in V_n$ is enumerable, i.e., there is an algorithm which produces the (possibly infinite) list of all pairs $\langle x, n\rangle$ such that $x \in V_n$. Then there is a constant c such that all elements of V_n have complexity at most $n + c$ (for any n).

(e) The "typical" binary string of length n has complexity close to n: there exists a constant c such that for any n more than 99% of all strings of length n have complexity in between $n - c$ and $n + c$.

Proof. (a) The asymptotically optimal decompression algorithm U is not worse that the trivial decompression algorithm $V(y) = y$.

(b) The number of such x's does not exceed the number of their compressed versions, which is limited by the number of all binary strings of length not exceeding n, which is bounded by 2^{n+1}. On the other hand, the number of x's such that $K(x) \leqslant n$ is not less than 2^{n-c} (here c is the constant from (a)), because all words of length $n - c$ have complexity not exceeding n.

(c) Let U be the optimal decompression algorithm used in the definition of K. Compare U with decompression algorithm $V : y \mapsto f(U(y))$:

$$K_U(f(x)) \leqslant K_V(f(x)) + O(1) \leqslant K_U(x) + O(1)$$

(any U-compressed version of x is a V-compressed version of $f(x)$).

(d) We allocate strings of length n to be compressed versions of strings in V_n (when a new element of V_n appears during the enumeration, the first unused string of length n is allocated). This procedure provides a decompression algorithm W such that $K_W(x) \leqslant n$ for any $x \in V_n$.

(e) According to (a), all the 100% of strings of length n have complexity not exceeding $n+c$ for some c. It remains to mention that the number of strings whose complexity is less than $n-c$ does not exceed the number of all their descriptions, i.e., strings of length less than $n-c$. Therefore, for $c = 7$ the fraction of strings having complexity less than $n-c$ among all the strings of length n does not exceed 1%.

Problems

1. A decompression algorithm D is chosen in such a way that $K_D(x)$ is even for any string x. Could D be optimal?

2. The same question if $K_D(x)$ is a power of 2 for any x.

3. Let D be the optimal decompression algorithm. Does it guarantee that $D(D(x))$ is also an optimal decompression algorithm?

4. Let $D_1, D_2, \ldots$ be a computable sequence of decompression algorithms. Prove that $K(x) \leqslant K_{D_i}(x) + 2\log i + O(1)$ for all i and x (the constant in $O(1)$ does not depend on x and i).

5.* Is it true that $K(xy) \leqslant K(x) + K(y) + O(1)$ for all x and y?

6 Algorithmic properties of K

Theorem 2 *The complexity function K is not computable; moreover, any computable lower bound for K is bounded from above.*

Proof. Assume that k is a computable lower bound for K which is not bounded from above. Then for any m we can effectively find a string x such that $K(x) > m$ (indeed, we should compute $k(x)$ for all strings x until we find a string x such that $k(x) > m$). Now consider the function

$$f(m) = \text{the first string } x \text{ such that } k(x) > m$$

Here "first" means "first discovered" and m is a natural number written in binary notation. By definition, $K(f(m)) > m$; on the other hand, f is a computable function and therefore $K(f(m)) \leqslant K(m) + O(1)$. But $K(m) \leqslant |m| + O(1)$, so we get that $m \leqslant |m| + O(1)$ which is impossible (the left-hand side is a natural number, the right-hand side—the length of its binary representation).

This proof is a formal version of the well-known paradox about "the smallest natural number which cannot be defined by twelve English words" (the quoted sentence defines this number and contains exactly twelve words).

7 Complexity and incompleteness

The argument used in the proof of the last theorem may be used to obtain an interesting version of Gödel incompleteness theorem. This application of complexity theory was advertised by Chaitin.

Consider a formal theory (like formal arithmetic or formal set theory). It may be represented as a (non-terminating) algorithm which generates statements of some fixed formal language; generated statements are called *theorems*. Assume that the language is rich enough to contain statements like "complexity of 010100010 is bigger than 765" (for any bit string and any natural number). The language of formal arithmetic satisfies this condition as well as the language of formal set theory. Let us assume also that all theorems are true.

Theorem 3 *There exists a constant c such that all the theorems of type "$K(x) > n$" have $n < c$.*

Indeed, assume that it is not true. Consider the following algorithm α: For a given integer k, generate all the theorems and look for a theorem of type $K(x) > s$ for some x and some s greater than k. When such a theorem is found, x becomes the output $\alpha(s)$ of the algorithm. By our assumption, $\alpha(s)$ is defined for all s.

All theorems are supposed to be true, therefore $\alpha(s)$ is a bit string whose complexity is bigger than s. As we have seen, this is impossible, since $K(\alpha(s)) \leqslant K(s) + O(1) \leqslant |s| + O(1)$ where $|s|$ is the length of the binary representation of s. (End of proof.)

(We may also use the statement of the preceding theorem instead of repeating the proof.)

Such a constant c can be found explicitly if we fix a formal theory and the optimal decompression algorithm and for most natural choices does not exceed — to give a rough estimate — $100,000$. It leads to a paradoxical situation: Toss a coin 10^6 times and write down the bit string of length $1,000,000$. Then with overwhelming probability its complexity will be bigger than $100,000$ but this claim will be unprovable in formal arithmetic or set theory. (The existence of true unprovable statement constitutes the well-known Gödel incompleteness theorem.)

8 Algorithmic properties of K (continued)

Theorem 4 *The function $K(x)$ is "enumerable from above", i.e., $K(x)$ can be represented as $\lim\limits_{n\to\infty} k(x,n)$ where $k(x,n)$ is a total computable function with integer values and*

$$k(x,0) \geqslant k(x,1) \geqslant k(x,2) \geqslant \ldots$$

Note that all values are integers, so for any x there exist some N such that $k(x,n) = K(x)$ for any $n > N$.

Proof. Let $k(x,n)$ be the complexity of x if we restrict by n the computation time used for decompression. In other words, let U be the optimal decompression algorithm used in the definition of K. Then $k(x,n)$ is the minimal $|y|$ for all y such that $U(y) = x$ and the computation time for $U(y)$ does not exceed n. (End of proof.)

(Technical correction: it can happen (for small n) that our definition gives $k(x,n) = \infty$. In this case we let $k(x,n) = |x| + c$ where c is chosen in such a way that $K(x) \leqslant |x| + c$ for any x.)

Therefore, any optimal decompression algorithm U is not everywhere defined (otherwise K_U would be computable). It sounds like a paradox: If $U(x)$ is undefined for some x we can extend U on x and let $U(x) = y$ for some y; after that $K_U(y)$ becomes smaller. However, it can be done for one x or for finite number of x's but we cannot make U defined everywhere and keep U optimal at the same time.

9 An encodings-free definition of complexity

The following theorem provides an "encodings-free" definition of Kolmogorov complexity as a minimal function K such that K is enumerable from above and $|\{x \mid K(x) < n\}| = O(2^n)$.

Theorem 5 *Let $K'(x)$ be any enumerable from above function such that $|\{x \mid K'(x) < n\}| \leqslant C2^n$ for some constant C and for all n. Then there exists a constant c such that $K(x) \leqslant K'(x) + c$ for all x.*

Proof. This theorem is a reformulation of one of the statements above. Let V_n be the set of all strings such that $K'(x) < n$. The binary relation $x \in V_n$ (between x and n) is enumerable. Indeed, $K'(x) = \lim k'(x,m)$ where k' is a total computable function that is decreasing as a function of m. Compute $k'(x,m)$ for all x and m in parallel. If it happens that $k(x,m) < n$ for some x and m, add x into the enumeration of V_n. (The monotonicity of k' guarantees that in this case $K'(x) < n$.) Since $\lim k'(x,m) = K'(x)$, any element of V_n will ultimately appear.

By our assumption $|V_n| \leqslant C2^n$. Therefore we can allocate strings of length $n+c$ (where $c = \lceil \log_2 C \rceil$) as descriptions of elements of V_n and will not run out of descriptions. So we get a decompression algorithm D such that $K_D(x) \leqslant n + c$ for $x \in V_n$. Since $K'(x) < n$ implies $K_D(x) \leqslant n + c$ for any x and n, we have $K_D(x) \leqslant K'(x) + 1 + c$ and $K(x) \leqslant K'(x) + c$ for some other c and any x. (End of proof.)

10 Axioms of complexity

It would be nice to have a list of "axioms" for Kolmogorov complexity that determine it uniquely (up to a bounded term). The following list shows one of the possibilities.

A1. (Conservation of information) For any computable (partial) function f there exists a constant c such that $\mathsf{K}(f(x)) \leqslant \mathsf{K}(x) + c$ for all x such that $f(x)$ is defined.

A2. (Enumerability from above) Function K is enumerable from above.

A3. (Units of measure) There are constants c and C such that the cardinality of set $\{x \mid \mathsf{K}(x) < n\}$ lies in between $c \cdot 2^n$ and $C \cdot 2^n$.

Theorem 6 *Any function K that satisfies A1–A3 differs from K only by $O(1)$ additive term.*

Proof. Axioms A2 and A3 guarantee that $K(x) \leqslant \mathsf{K}(x) + O(1)$ (here K is any function satisfying the axioms, while K is Kolmogorov complexity). We need to prove that $\mathsf{K}(x) \leqslant K(x) + O(1)$.

First, we prove that $\mathsf{K}(x) \leqslant |x| + O(1)$.

Since K is enumerable from above, we can generate strings x such that $\mathsf{K}(x) < n$. Axiom A3 guarantees that we have at least 2^{n-d} strings with this property for some d (which we assume to be integer). Let us stop generating them when we have already 2^{n-d} strings x such that $\mathsf{K}(x) < n$; let S_n be the set of strings generated in this way. The list of all elements in S_n can be obtained by an algorithm that has n as input; $|S_n| = 2^{n-d}$ and $\mathsf{K}(x) < n$ for any $x \in S_n$.

We may assume that $S_1 \subset S_2 \subset S_3 \subset \ldots$ (if not, replace some elements of S_i by elements of S_{i-1} etc.). Let T_i be equal to $S_{i+1} \setminus S_i$. Then T_i has 2^{n-d} elements and all T_i are disjoint.

Now consider a computable function f that maps elements of T_n onto strings of length $n-d$. Axiom A1 guarantees then that $\mathsf{K}(x) = n + O(1)$ for any string of length $n-d$. Therefore, $\mathsf{K}(x) \leqslant |x| + O(1)$ for all x.

Let D be the optimal decompression algorithm. We apply A1 to function D. If p is a shortest description for x, then $D(x) = p$, therefore $\mathsf{K}(x) = \mathsf{K}(D(p)) \leqslant \mathsf{K}(p) + O(1) \leqslant |p| + O(1) = K(x) + O(1)$.

Problems

1. If $f : \mathbb{N} \to \mathbb{N}$ is a computable bijection, then $K(f(x)) = K(x) + O(1)$. Is it true if f is a (computable) injection (i.e., $f(x) \neq f(y)$ for $x \neq y$)? Is it true if f is a surjection (for any y there is an x such that $f(x) = y$)?

2. Prove that $K(x)$ is "continuous" in the following sense: $K(x0) = K(x) + O(1)$ and $K(x1) = K(x) + O(1)$.

3. Is it true that $K(x)$ changes at most by a constant if we change the first bit in x? last bit in x? any bit in x?

4. Prove that $K(\bar{x}01\,\mathrm{bin}(K(x)))$ (a string x with doubled bits is concatenated with 01 and the binary representation of its complexity $K(x)$) equals $K(x) + O(1)$.

11 Complexity of pairs

Let

$$x, y \mapsto [x, y]$$

be any computable function which maps pairs of strings into strings and is an injection (i.e., $[x,y] \neq [x',y']$ if $x \neq x'$ or $y \neq y'$). We define complexity $K(x,y)$ of pair of strings as $K([x,y])$.

Note that $K(x,y)$ changes only by $O(1)$-term if we consider another computable "pairing function": If $[x,y]_1$ and $[x,y]_2$ are two pairing functions, then $[x,y]_1$ can be obtained from $[x,y]_2$ by an algorithm, so $K([x,y]_1) \leqslant K([x,y]_2) + O(1)$.

Note that

$$K(x,y) \geqslant K(x) \quad \text{and} \quad K(x,y) \geqslant K(y)$$

(indeed, there are computable functions that produce x and y from $[x,y]$).

For similar reasons, $K(x,y) = K(y,x)$ and $K(x,x) = K(x)$.

We can define $K(x,y,z)$, $K(x,y,z,t)$ etc. in a similar way: $K(x,y,z) = K([x,[y,z]])$ (or $K(x,y,z) = K([[x,y],z])$, the difference is $O(1)$).

Theorem 7

$$K(x,y) \leqslant K(x) + 2\log K(x) + K(y) + O(1).$$

Proof. By $\bar{x}$ we denote binary string x with all bits doubled. Let D be the optimal decompression algorithm. Consider the following decompression algorithm D_2:

$$\overline{\mathrm{bin}(|p|)}01pq \mapsto [D(p), D(q)].$$

Note that D_2 is well defined, because the input string $\overline{\mathrm{bin}(|p|)}01pq$ can be disassembled into parts uniquely: we know where 01 is, so we can find $|p|$ and then separate p and q.

If p is the shortest description for x and q is the shortest description for y, then $D(p) = x$, $D(q) = y$ and $D_2(\overline{\mathrm{bin}(p)}01pq) = [x,y]$. Therefore

$$K_{D_2}([x,y]) \leqslant |p| + 2\log|p| + |q| + O(1);$$

here $|p| = K(x)$ and $|q| = K(y)$ by our assumption. (End of proof.)

Of course, p and q can be interchanged: we can replace $\log K(p)$ by $\log K(q)$.

12 Conditional complexity

We now want to define conditional complexity of x when y is known. Imagine that you want to send string x to your friend using as few bits as possible. If she already knows some string y which is similar to x, this can be used.

Here is the definition. Let $\langle p,y\rangle \mapsto D(p,y)$ be a computable function of two arguments. We define conditional complexity $K_D(x|y)$ of x when y is known as

$$K_D(x|y) = \min\{\,|p| \mid D(p,y) = x\}.$$

As usual, $\min(\varnothing) = +\infty$. The function D is called "conditional decompressing function" or "conditional description mode": p is the description (compressed version) of x when y is known. (To get x from p the decompressing algorithm D needs y.)

Theorem 8 *There exists an optimal conditional decompressing function D such that for any other conditional decompressing function D' there exists a constant c such that*

$$K_D(x|y) \leqslant K_{D'}(x|y) + c$$

for any strings x and y.

Proof is similar to the proof of unconditional theorem. Consider some programming language where programs allow two input strings and are self-delimiting. Then let

$$D(uv, y) = \text{the output of program } u \text{ applied to } v, y.$$

Algorithm D finds a (self-delimiting) program u as a prefix of its first argument and then applies u to the rest of the first argument and the second argument.

Let D' be any other conditional decompressing function. Being computable, it has some program u. Then

$$K_D(x|y) \leqslant K_{D'}(x|y) + |u|.$$

Indeed, let p be the shortest string such that $D'(p, y) = x$ (therefore, $|p| = K_{D'}(x|y)$). Then $D(up, y) = x$, therefore $K_D(x|y) \leqslant |up| = |p| + |u| = K_{D'}(x|y) + |u|$. (End of proof.)

We fix some optimal conditional decompressing function D and omit index D in $K_D(x|y)$. Beware that $K(x|y)$ is defined only "up to $O(1)$-term".

Theorem 9 *(a) $K(x|y) \leqslant K(x) + O(1)$*

(b) For any fixed y there exists some constant c such that

$$|K(x) - K(x|y)| \leqslant c.$$

This theorem says that conditional complexity is smaller than unconditional one but for any fixed condition the difference is limited by a constant (depending on the condition).

Proof. (a) If D_0 is an (unconditional) decompressing algorithm, we can consider a conditional decompressing algorithm

$$D(p, y) = D_0(p)$$

that ignores conditions. Then $K_D(x|y) = K_{D_0}(x)$.

(b) On the other hand, if D is a conditional decompressing algorithm, for any fixed y we can consider an (unconditional) decompressing algorithm D_y defined as

$$D_y(p) = D(p, y).$$

Then $K_{D_y}(x) = K_D(x|y)$ for given y and for all x. And $K(x) \leqslant K_{D_y}(x) + O(1)$ (where $O(1)$-constant depends on y). (End of proof.)

13 Pair complexity and conditional complexity

Theorem 10

$$K(x, y) = K(x|y) + K(y) + O(\log K(x) + \log K(y)).$$

Proof. Let us prove first that

$$K(x,y) \leqslant K(x|y) + K(y) + O(\log K(x) + \log K(y)).$$

We do it as before: If D is an optimal decompressing function (for unconditional complexity) and D_2 is an optimal conditional decompressing function, let

$$D'(\overline{\mathrm{bin}(p)}01pq) = [D_2(p, D(q)), D(q)].$$

In other terms, to get the description of pair x, y we concatenate the shortest description of y (denoted by q) with the shortest description of x when y is known (denoted by p). (Special precautions are used to guarantee the unique decomposition.) Indeed, in this case $D(q) = y$ and $D_2(p, D(q)) = D_2(p, y) = x$, therefore

$$K_{D'}([x,y]) \leqslant |p| + 2\log|p| + |q| + O(1) \leqslant$$
$$\leqslant K(x|y) + K(y) + O(\log K(x) + \log K(y)).$$

The reverse inequality is much more interesting. Let us explain the idea of the proof. This inequality is a translation of a simple combinatorial statement. Let A be a finite set of pairs of strings. By $|A|$ we denote the cardinality of A. For any string y we consider the set A_y defined as

$$A_y = \{x | \langle x, y\rangle \in A\}.$$

The cardinality $|A_y|$ depends on y (and is equal to 0 for all y outside some finite set). Evidently,

$$\sum_y |A_y| = |A|.$$

Therefore, the number of y such that $|A_y|$ is big, is limited:

$$|\{y | \, |A_y| \geqslant c\}| \leqslant |A|/c$$

for any c.

Now we return to complexities. Let x and y be two strings. The inequality $K(x|y) + K(y) \leqslant K(x,y) + O(\log K(x) + \log K(y))$ can be informally read as follows: if $K(x,y) < m + n$, then either $K(x|y) < m$ or $K(y) < n$ up to logarithmic terms. Why it is the case? Consider a set A of all pairs $\langle x, y\rangle$ such that $K(x,y) < m + n$. There is at most 2^{m+n} pairs in A. The given pair $\langle x, y\rangle$ belongs to A. Consider the set A_y. It is either "small" (contains at most 2^m elements) or big. If A_y is small ($|A_y| \leqslant 2^m$), then x can be described (when y is known) by its ordinal number in A_y, which requires m bits, and $K(x|y)$ does not exceed m (plus some administrative overhead). If A_y is big, then y belongs to a (rather small) set Y of all strings y such that A_y is big. The number of strings y such that $|A_y| > 2^m$ does not exceed $|A|/2^m = 2^n$. Therefore, y can be (unconditionally) described by its ordinal number in Y which requires n bits (plus overhead of logarithmic size).

Let us repeat this more formally. Let $K(x,y) = a$. Consider the set A of all pairs $\langle x, y\rangle$ that have complexity at most a. Let $b = \lfloor \log_2 |A_y| \rfloor$. To describe x when y is known we need to specify a, b and the ordinal number of x in A_y (this set can be enumerated effectively if a and b are known since K is enumerable from above). This ordinal number has $b + O(1)$ bits and, therefore, $K(x|y) \leqslant b + O(\log a + \log b)$.

On the other hand, the set of all y' such that $|A_{y'}| \geqslant 2^b$ consists of at most $|A|/2^b = O(2^{a-b})$ elements and can be enumerated when a and b are known. Our y belongs to this set, therefore, y can be described by a, b and y's ordinal number and $K(y) \leqslant a - b + O(\log a + \log b)$. Therefore, $K(y) + K(x|y) \leqslant a + O(\log a + \log b)$. (End of proof.)

Problems

1. Define $K(x,y,z)$ as $K([[x,y],[x,z]])$. Is this definition equivalent to a standard one (up to $O(1)$-term)?

2. Prove that $K(x,y) \leqslant K(x) + \log K(x) + 2\log\log K(x) + K(y) + O(1)$. (Hint: repeat the trick with encoded length.)

3. Let f be a computable function of two arguments. Prove that $K(f(x,y)|y) \leqslant K(x|y) + O(1)$ where $O(1)$-constant depends on f but not on x and y.

4**. Prove that $K(x|K(x)) = K(x) + O(1)$.

14 Applications of conditional complexity

Theorem 11 *If x, y, z are strings of length at most n, then*

$$2K(x,y,z) \leqslant K(x,y) + K(x,z) + K(y,z) + O(\log n)$$

Proof. The statement does not mention conditional complexity; however, the proof uses it. Recall that (up to $O(\log n)$-terms) we have

$$K(x,y,z) - K(x,y) = K(z|x,y)$$

and

$$K(x,y,z) - K(x,z) = K(y|x,z)$$

Therefore, our inequality can be rewritten as

$$K(z|x,y) + K(y|x,z) \leqslant K(y,z),$$

and the right-hand side is (up to $O(\log n)$) equal to $K(z|y) + K(y)$. It remains to note that $K(z|x,y) \leqslant K(z|y)$ (the more we know, the smaller is the complexity) and $K(y|x,z) \leqslant K(y)$. (End of proof.)

15 Incompressible strings

The string x of length n is called *incompressible* if $K(x|n) \geqslant n$. More liberal definition: x is c-incompressible, if $K(x|n) \geqslant n - c$.

Theorem 12 *For each n there exist incompressible strings of length n. For each n and each c the fraction of c-incompressible strings (among all strings of length n) is greater than $1 - 2^{-c}$.*

Proof. The number of descriptions of length less than $n - c$ is $1 + 2 + 4 + \ldots + 2^{n-c-1} < 2^{n-c}$. Therefore, the fraction of c-compressible strings is less than $2^{n-c}/2^n = 2^{-c}$. (End of proof.)

16 Computability and complexity of initial segments

Theorem 13 *An infinite sequence $x = x_1x_2x_3\ldots$ of zeros and ones is computable if and only if $K(x_1\ldots x_n|n) = O(1)$.*

Proof. If x is computable, then the initial segment $x_1\ldots x_n$ is a computable function of n, and $K(f(n)|n) = O(1)$ for any computable function of n.

Another direction is much more complicated. We provide this proof since is uses some methods typical for the general theory of computation (recursion theory).

Now assume that $K(x_1 \ldots x_n|n) < c$ for some c and all n. We have to prove that the sequence $x_1 x_2 \ldots$ is computable. A string of length n is called "simple" if $K(x|n) < c$. There is at most 2^c simple strings of any given length. The set of all simple strings is enumerable (we can generate them trying all short descriptions in parallel for all n).

We call a string "good" if all its prefixes (including the string itself) are simple. The set of all good strings is also enumerable. (Enumerating simple strings, we can select strings whose prefixes are found to be simple.)

Good strings form a subtree in full binary tree. (Full binary tree is a set of all binary strings. A subset T of full binary tree is a subtree if all prefixes of any string $t \in T$ are elements of T.)

The sequence $x_1 x_2 \ldots$ is an infinite branch of the subtree of good strings. Note that this subtree has at most 2^c infinite branches because each level has at most 2^c vertices.

Imagine for a while that subtree of good strings is decidable. (In fact, it is not the case, and we will need additional construction.) Then we can apply the following

Lemma 1. If a decidable subtree has only finite number of infinite branches, all these branches are computable.

Proof. If two branches in a tree are different then they diverge at some point and never meet again. Consider a level N where all infinite branches diverge. It is enough to show that for each branch there is an algorithm that chooses the direction of branch (left or right, i.e., 0 or 1) above level N. Since we are above level N, the direction is determined uniquely: if we choose a wrong direction, no infinite branches are possible. Compactness argument says that in this case a subtree rooted in the "wrong" vertex will be finite. This fact can be discovered at some point (recall that subtree is assumed to be decidable). Therefore, at each level we can wait until one of two possible directions is closed, and chose another one. This algorithm works only above level N, but the initial segment can be a compiled-in constant. (Lemma is proved.)

Application of Lemma 1 is made possible by the following

Lemma 2. Let G be a subtree of good strings. Then there exists a decidable subtree $G' \subset G$ that contains all infinite branches of G.

Proof. For each n let $g(n)$ be the number of good strings of length n. Consider an integer $g = \limsup g(n)$. In other words, there exist infinitely many n such that $g(n) = g$ but only finitely many n such that $g(n) > g$. We choose some N such that $g(n) \leqslant g$ for all $n \geqslant N$ and consider only levels $N, N+1, \ldots$

A level $n \geqslant N$ is called complete if $g(n) = g$. By our assumption there are infinitely many complete levels. On the other hand, the set of all complete levels is enumerable. Therefore, we can construct a computable increasing sequence $n_1 < n_2 < \ldots$ of complete levels. (To find n_{i+1}, we enumerate complete levels until we find $n_{i+1} > n_i$.)

There is an algorithm that for each i finds the list of all good strings of length n_i. (It waits until g goods strings of length n_i appear) Let us call all those strings (for all i) "selected". The set of all selected strings is decidable. If a string of length n_j is selected, then its prefix of length n_i (for $i < j$) is selected. It is easy to see now that selected strings and their prefixes form a decidable subtree G' that includes any infinite branch of G.

Lemma 2 and theorem 13 are proved.

For computable sequence $x_1 x_2 \ldots$ we have $K(x_1 \ldots x_n|n) = O(1)$ and therefore $K(x_1 \ldots x_n) \leqslant \log n + O(1)$. One can prove that the inequality also implies computability (see Problems). However, the inequality $K(x_1 \ldots x_n) = O(\log n)$ does not imply computability of $x_1 x_2 \ldots$.

Theorem 14 *Let A be any enumerable set of natural numbers. Then for its characteristic sequence $a_0 a_1 a_2 \ldots$ ($a_i = 1$ if $i \in A$ and $a_i = 0$ otherwise) we have*

$$K(a_0 a_1 \ldots a_n) = O(\log n)$$

Proof. To specify $a_0 \ldots a_n$ it is enough to specify two numbers. The first is n and the second is the number of 1's in $a_0 \ldots a_n$, i.e., the cardinality of the set $A \cap [0,n]$. Indeed, for a given n, we can enumerate this set, and since we know its cardinality, we know where to stop the enumeration. Both of them use $O(\log n)$ bits. (End of proof.)

This theorem shows that initial segments of characteristic sequences of enumerable sets are far from being incompressible.

As we know that for each n there exists an incompressible sequence of length n, it is natural to ask whether there is an infinite sequence $x_1 x_2 \ldots$ such that its initial segment of any length n is incompressible (or at least c-incompressible for some c that does not depend on n). The following theorem shows that it is not the case.

Theorem 15 *There exists c such that for any sequence $x_1 x_2 x_2 \ldots$ there are infinitely many n such that*

$$K(x_1 x_2 \ldots x_n) \leqslant n - \log n + c$$

Proof. The main reason why it is the case is that the series $\sum(1/n)$ diverges. It makes possible to select the sets $A_1, A_2, \ldots$ with following properties:

(1) each A_i consists of strings of length i;

(2) $|A_i| \leqslant 2^i/i$;

(3) for any infinite sequence $x_1 x_2 \ldots$ there are infinitely many i such that $x_1 \ldots x_i \in A_i$.

(4) the set $A = \cup_i A_i$ is decidable.

Indeed, starting with some A_i, we cover about $(1/i)$-fraction of the whole space Ω of all infinite sequences. Then we can choose A_{i+1} to cover other part of Ω, and so on until we cover all Ω (it happens because $1/i + 1/(i+1) + \ldots + 1/j$ goes to infinity). Then we can start again, providing a second layer of covering, etc.

It is easy to see that $|A_1| + |A_2| + \ldots + |A_i| = O(2^i/i)$: Each term is almost twice as big as the preceding one, therefore, the sum is $O(\text{last term})$. Therefore, if we write down in lexicographic ordering all the elements of $A_1, A_2, \ldots$, any element x of A_i will have number $O(2^i/i)$. This number determines x uniquely and therefore for any $x \in A_i$ we have

$$K(x) \leqslant \log(O(2^i)/i) = i - \log i + O(1).$$

. (End of proof.)

Problems

1. True or false: for any computable function f there exists a constant c such that $K(x|y) \leqslant K(x|f(y)) + c$ for all x, y such that $f(y)$ is defined.

2. Prove that $K(x_1 \ldots x_n|n) \leqslant \log n + O(1)$ for any characteristic sequence of an enumerable set.

3*. Prove that there exists a sequence $x_1 x_2 \ldots$ such that $K(x_1 \ldots x_n) \geqslant n - 2\log n - c$ for some c and for all n.

4*. Prove that if $K(x_1 \ldots x_n) \leqslant \log n + c$ for some c and all n, then the sequence $x_1 x_2 \ldots$ is computable.

17 Incompressibility and lower bounds

We apply Kolmogorov complexity to obtain lower bound for the following problem. Let M be a Turing machine (with one tape) that duplicates its input: for any string x on the tape (with blanks on the right of x) it produces xx. We prove that M requires time $\Omega(n^2)$ if x is an incompressible string of length n. The idea is simple: the head of TM can carry finite number of bits with limited speed, therefore the speed of information transfer (measured in bit$\times$cell/step) is bounded and to move n bits by n cells we need $\Omega(n^2)$ steps.

Theorem 16 *Let M be any Turing machine. Then there exists some constant c with the following property: for any k, any $l \geqslant k$ and any t, if cells c_i with $i > k$ are initially empty, then the complexity of the string $c_{l+1}c_{l+2}\ldots$ after t steps is bounded by $ct/(l-k)+O(\log l+\log t)$.*

Roughly speaking, if we have to move information at least by $l-k$ cells, then we can bring at most $ct/(l-k)$ bits into the area where there was no information at the beginning.

One technical detail: string $s_{l+1}c_{l+2}\ldots$ denotes the visited part of the tape.

This theorem can be used to get a lower bound for duplication. Let x be an incompressible string of length n. We apply duplicating machine to the string $0^n x$ (with n zeros before x). After the machine terminates in t steps, the tape is $0^n x 0^n x$. Let $k = 2n$ and $l = 3n$. We can apply our theorem and get $n \leqslant K(x) \leqslant ct/n + O(\log n + \log t)$. Therefore, $t = \Omega(n^2)$ (note that $\log t < 2\log n$ unless $t > n^2$).

Proof. Let u be any point on the tape between k and l. A police officer records what TM carries is its head while crossing point u from left to right (but not the time of crossing). The recorded sequence T_u of TM-states is called *trace* at point u. Each state occupies $O(1)$ bits since the set of states is finite. This trace together with u, k, l and the number of steps after the last crossing (at most t) is enough to reconstruct the contents of $c_{l+1}c_{l+2}\ldots$ at the moment t. (Indeed, we can simulate the behaviour of M on the right of u.) Therefore, $K(c_{l+1}c_{l+2}\ldots) \leqslant cN_u + O(\log l) + O(\log t)$ where N_u is the length of T_u, i.e., the number of crossings at u.

Now we add these inequalities for all $u = k, k+1, \ldots, l$. The sum of N_u is bounded by t (since only one crossing is possible at any given time). So

$$(l-k)K(c_{l+1}c_{l+2}\ldots) \leqslant t + (l-k)[O(\log l) + O(\log t)]$$

and theorem is proved.

The original result (one of the first lower bounds for time complexity) was not for duplication but for palindrome recognition: Any TM that checks whether its input is a palindrome (like `abadaba`) needs $\Omega(n^2)$ steps for some inputs of length n. We can prove it by incompressibility method.

Proof sketch: Consider a palindrome xx^R of length $2n$. Let u be any position in the first half of xx^R: $x = yz$ and length of y is u. Then the trace T_u determines y uniquely if we record states of TM while crossing checkpoint u in both directions. Indeed, if strings with different y have the same trace, we can mix the left part of one computation with the right part of another one and get a contradiction. Taking all u between $|x|/4$ and $|x|/2$, we get the required bound.

18 Incompressibility and prime numbers

Let us prove that there are infinitely many prime numbers. Imagine that there are only n prime numbers $p_1, \ldots, p_n$. Then each integer N can be factored as

$$N = p_1^{k_1} p_2^{k_2} \ldots p_n^{k_n}.$$

where all k_i do not exceed $\log N$. Therefore, each N can be described by n integers $k_1, \ldots, k_n$ and $k_i \leqslant \log N$ for any i, so the total number of bits needed to describe N is $O(n \log\log N)$. But integer N corresponds to a string of length $\log N$, so we get a contradiction if this string is incompressible.

19 Incompressible matrices

Consider an incompressible Boolean matrix of size $n \times n$. Let us prove that its rank (over field $\mathbb{F}_2 = \{0,1\}$) is greater than $n/2$.

Indeed, imagine that its rank is at most $n/2$. Then we can select $n/2$ columns of the matrix such that any other column is a linear combination of selected ones. Let $k_1, \ldots, k_{n/2}$ be the numbers of these columns.

Then instead of specifying all bits of the matrix we can specify:

(1) the numbers $k_1, \ldots, k_n$ ($O(n \log n)$ bits)

(2) bits in the selected columns ($n^2/2$ bits)

(3) $n^2/4$ bits that are coefficients in linear combinations of selected columns needed to get any non-selected column, ($n/2$ bits for any of $n/2$ non-selected columns).

Therefore, we get $0.75n^2 + O(n \log n)$ bits instead of n^2 needed for incompressible matrix.

Of course, it is trivial to find a $n \times n$ Boolean matrix of full rank, so why this construction is useful? In fact, the same idea shows that incompressible matrix has minors of big rank (see LV for details).

20 Incompressible graphs

Any graph with n vertices can be represented by a bit string of length $n(n-1)/2$. We call a graph *incompressible* if this string is incompressible.

Let us show that incompressible graph is connected. Indeed, imagine that it can be divided into two connected components, and one of them (smaller) has k vertices ($k < n/2$). Then the graph can be described by

(1) numbers of k vertices in this component ($k \log n$ bits)

(2) $k(k-1)/2$ and $(n-k)(n-k-1)/2$ bits needed to describe both components.

In (2) (compared to the full description of the graph) we save $k(n-k)$ bits for edges that go from one component to another one, and $k(n-k) > O(k \log n)$ for big enough n (recall that $k < n/2$).

21 Incompressible tournaments

Let M be a tournament, i.e., a complete directed graph with n vertices (for any two different vertices i and j there exists either edge $i \to j$ or $j \to i$ but not both).

A tournament is *transitive* if vertices are linearly ordered by the relation $i \to j$.

Lemma. Each tournament of size $2^k - 1$ has a transitive sub-tournament of size k.

Proof. (Induction by n) Let x be any vertex. Then $2^k - 2$ remaining vertices are divided into two groups: "smaller" than x and "greater" than x. At least one of the groups has $2^{k-1} - 1$ elements and contains transitive sub-tournament of size $k-1$. Adding x to it, we get a transitive sub-tournament of size k.

This lemma gives a lower bound on the size of graph that does not include transitive k-tournament.

The incompressibility method provides an upper bound: an incompressible tournament with n vertices may have transitive sub-tournaments of $O(\log n)$ size only.

A tournament with n vertices is represented by $n(n-1)/2$ bits. If a tournament R with n vertices has transitive sub-tournament R' of size k, then R can be described by:

(1) numbers of vertices in R' listed according to linear R'-ordering ($k \log n$ bits)

(2) remaining bits in the description of R (except for bits that describe relations inside R')

In (2) we save $k(k-1)/2$ bits, and in (1) we use $k \log n$ additional bits. Since we have to lose more than we win, $k = O(\log n)$.

22 Discussion

All these results can be considered as direct reformulation of counting (or probabilistic arguments). Moreover, counting gives us better bounds without $O()$-notation.

But complexity arguments provide an important heuristics: We want to prove that random object x has some property and note that if x does not have it, then x has some regularities that can be used to give a short description for x.

Problems

1. Let x be an incompressible string of length n and let y be a longest substring of x that contains only zeros. Prove that $|y| = O(\log n)$

2^*. Prove that $|y| = \Omega(\log n)$.

3. (LV, 6.3.1) Let $w(n)$ be the largest integer such that for each tournament T on $N = \{1, \ldots, n\}$ there exist disjoint sets A and B, each of cardinality $w(n)$, such that $A \times B \subseteq T$. Prove that $w(n) \leqslant 2\lceil \log n \rceil$. (Hint: add $2w(n)\lceil \log n \rceil$ bit to describe nodes, and save $w(n)^2$ bits on edges. Source: P. Erdös and J. Spencer, Probabilistic methods in combinatorics, Academic Press, 1974.)

23 k- and $k+1$-head automata

A k-head finite automaton has k (numbered) heads that scan from left to right an input string (which is the same for all heads). Automaton has a finite number of states. Transition table specifies an action for each state and each k-tuple of input symbols. Action includes new state and the set of heads to be moved. (We may assume that at least one head should be moved; otherwise we can precompute the transition.)

One of the states is called an *initial* state. Some states are *accepting* states. An automaton A accepts string x if A comes to an accepting state after reading x starting from the initial state. (Reading x is finished when all heads leave x. We require that this happens for any string x.)

For $k = 1$ we get the standard notion of finite automaton.

Example: A 2-head automaton can recognize strings of form $x\#x$ (where x is a binary string). The first head moves to #-symbol and then both heads move and check whether they see the same symbols.

It is well known that this language cannot be recognized by 1-head finite automaton, so 2-head automata are more powerful that 1-head ones.

Our goal is to prove separation for bigger k.

Theorem 17 *For any k there exists a language that can be recognized by $(k+1)$-head automaton but not by k-head one.*

The language is similar to the language considered above. For example, for $k = 2$ we consider a language consisting of strings

$$x\#y\#z\#z\#y\#x$$

Using three heads, we can easily recognize this language. Indeed, the first head moves from left to right and ignores the left part of the input string. While it reaches (right) y, another head is used to check whether y on the left coincides with y on the right. (The first head waits till the second one crosses x and reaches y.) When the first head then reaches x, the third head is used to check x. After that the first head is of no use, but second and third heads can be used to check z.

The same approach shows that an automaton with k heads can recognize language L_N that consists of strings

$$x_1\#x_2\#\ldots\#x_N\#x_N\#\ldots\#x_2\#x_1$$

for $N = (k-1) + (k-2) + \ldots + 1 = k(k-1)/2$ (and for all smaller N).

Let us prove now that k-head automaton A cannot recognize L_N if N is bigger than $k(k-1)/2$. In particular, no automaton with 2 heads recognizes L_3 and even L_2)

Let us fix a string

$$x = x_1\#x_2\#\ldots\#x_N\#x_N\#\ldots\#x_2\#x_1$$

where all x_i have the same length l and the string $x_1x_2\ldots x_N$ is an incompressible string (of length Nl). String x is accepted by A. In our argument the following notion is crucial: We say that (unordered) pair of heads "covers" x_m if at some point one head is inside the left instance of x_m while another head is inside the right instance.

After that the right head can visit only strings $x_{m-1},\ldots,x_1$ and left head cannot visit left counterparts of those strings (they are on the left of it). Therefore, only one x_m can be covered by any given pair of heads.

In our example we had three heads (and, therefore, three pairs of heads) and each string x_1,x_2,x_3 was covered by one pair.

The number of pairs is $k(k-1)/2$ for k heads. Therefore there exists some x_m that was not covered at all during the computation. We show that conditional complexity of x_m when all other x_i are known does not exceed $O(\log l)$. (The constant here depends on N and A, but not on l.) This contradicts to the incompressibility of $x_1\ldots x_N$ (we can replace x_m by self-delimiting description of x_m when other x_i are known and get a shorter description of incompressible string).

The bound for the conditional complexity of x_m can be obtained in the following way. During the accepting computation we take special care of the periods when one of the heads is inside x_m (any of two copies). We call there periods critical sections. Note that each critical section is either L-critical (some heads are inside left copy of x_m) or R-critical but not both (no pair of heads covers x_m). Critical section starts when one of the heads moves inside x_m (other heads can also move in during the section) and ends when all heads leave x_m. Therefore, the number of critical sections during the computation is at most $2k$.

Let us record the positions of all heads and the state of automaton at the beginning and at the end of each critical section. This requires $O(\log l)$ bits (note that we do not record time and may assume without loss of generality that heads do not move more than one cell out of the input string).

We claim that this information (called *trace* in the sequel) determines x_m if all other x_i are known. To see why, let us consider two computations with different x_m and x'_m but the same x_i for $i\neq m$ and the same traces.

Equal traces allow us to "cut and paste" these two computations on the boundaries of critical sections. (Outside the critical sections computations are the same, because the strings are identical except for x_m and state and positions after each critical section are included in a trace.) Now we take L-critical sections from one computation and R-critical sections from another one. We get a mixed computation that is an accepting run of A on a string that has x_m on the left and x'_m on the right. Therefore, A accepts string that it should not accept. (End of proof.)

24 Heap sort: time analysis

(This section assumes that you know what heapsort is.)

Let us assume that we sort numbers $1,2,\ldots,N$. We have $N!$ possible permutations. Therefore, to specify any permutation we need about $\log N!$ bits. Stirling formula says that $N!\approx(N/e)^N$, therefore the number of bits needed to specify one permutation is $N\log N+O(N)$. As usual, most of the permutations are incompressible in the sense that they have complexity at least $O(N\log N)-O(N)$. We estimate the number of operations for heap sort in case of incompressible permutation.

Heap sort consists of who phases. First phase creates a heap out of array. (The indexes in array $a[1..N]$ form a tree where $2i$ and $2i+1$ are sons of i. Heap property says that ancestor has bigger value that any of its descendants.)

Transforming array into a heap goes as follows: for each $i=N,N-1,\ldots,1$ we make the heap out of subtree rooted at i. Doing this for node i, we need $O(k)$ steps where k is the distance between node i and the leaves of the tree. Therefore, $k=0$ for about half of nodes, $k=1$ for about $1/4$ of nodes etc., the average number of steps per node is $O(\sum k2^{-k})=O(1)$, and the total number of operations is $O(N)$.

Important observation: after the heap is created, the complexity of array $a[1..N]$ is still $N\log N + O(N)$, if the initial permutation was incompressible. Indeed, heapifying means composition of initial permutation with some other permutation (which is determined by results of comparisons between array elements). Since total time for heapifying is $O(N)$, there are at most $O(N)$ comparisons and their results form a bit string of length $O(N)$ that determines the heapifying permutation. The initial (incompressible) permutation is a composition of the heap and $O(N)$-permutation, therefore heap has complexity at least $N\log N - O(N)$.

The second phase transforms heap into sorted array. At any stage array is divided into parts: $a[1..n]$ is still a heap, but $a[n+1..N]$ is the end of the sorted array. One step of transformation (it decreases n by 1) goes as follows: the maximal heap element $a[1]$ is taken out of the heap and exchanged with $a[n]$. Therefore, $a[n..N]$ is now sorted, and heap property is almost true: ascendant has bigger value that descendant unless ascendant is $a[n]$ (that is now in root position). To restore heap property, we move $a[n]$ down the heap. The question is how many steps do we need. If the final position is d_n levels above the leaves level, we need $\log N - d_n$ exchanges, and the total number of exchanges is $N\log N - \sum d_n$.

We claim that $\sum d_n = O(N)$ for incompressible permutation, and, therefore, the total number of exchanges is $N\log N + O(N)$. (There are different implementation of heapsort. A careful one first looks for the possible path for the new element, then looks for its position (starting from the leaves) and then actually moves new element, thus making only $N\log N + O(N)$ assignments and $2N\log N + O(N)$ comparisons. See LV for details.)

So why $\sum d_n$ is $O(N)$? Let us record the direction of movements while elements fall down through the heap (using 0 and 1 for left and right). We don't use delimiters to separate strings that correspond to different n and use $N\log N - \sum d_i$ bits altogether. Separately we write down all d_n in self-delimiting way. This requires $\sum(2\log d_i + O(1))$ bits. All this information allows us to reconstruct all moves during the second phase, and therefore to reconstruct initial state of the heap before the second phase. Therefore, the complexity of heap before the second phase (which is $N\log N - O(N)$) does not exceed $N\log N - \sum d_n + \sum(2\log d_n) + O(N)$, therefore, $\sum(d_n - 2\log d_n) = O(N)$. Since $2\log d_n < 0.5d_n$ for $d_n > 16$ (and all smaller d_n have sum $O(N)$ anyway), we conclude that $\sum d_n = O(N)$.

Problems

1^*. Prove that for most pairs of binary strings x, y of length n any common subsequence of x and y has length at most $0.99n$ (for large enough n).

25 Infinite random sequences

There is some intuitive feeling saying that a fair coin tossing cannot produce sequence

$$00000000000000000000000\ldots$$

or

$$01010101010101010101010\ldots,$$

therefore, infinite sequences of zeros and ones can be divided in two categories. Random sequences are sequences that can appear as the result of infinite coin tossing; non-random sequences (like two sequences above) cannot appear. It is more difficult to provide an example of a random sequence (it somehow becomes non-random after the example is provided), so our intuition is not very reliable here.

26 Classical probability theory

Let Ω be the set of all infinite sequences of zeros and ones. We define an *uniform Bernoulli measure* on Ω as follows. For each binary string x let Ω_x be the set of all sequences that have prefix x (a subtree

rooted at x).

Consider a measure P such that $P(\Omega_x) = 2^{-|x|}$. Lebesgue theory allows us to extend this measure to all Borel sets (and even farther).

A set $X \subset \Omega$ is called *null* set, if $P(X)$ is defined and $P(X) = 0$. Let us give a direct equivalent definition that is useful for constructive version:

A set $X \subset \Omega$ is a null set if for every $\varepsilon > 0$ there exists a sequence of binary strings $x_0, x_1, \ldots$ such that

(1) $X \subset \Omega_{x_0} \cup \Omega_{x_1} \cup \ldots$;

(2) $\sum_i 2^{-|x_i|} < \varepsilon$.

Note that $2^{-|x_i|}$ is $P(\Omega_{x_i})$ according to our definition. In words: X is a null set if it can be covered by a sequence of intervals Ω_{x_i} whose total measure is as small as we wish.

Examples: Each singleton is a null set. A countable union of null sets is a null set. A subset of a null set is a null set. The set Ω is not a null set (compactness). The set of all sequences that have zeros at positions with even numbers is a null set.

27 Strong Law of Large Numbers

Informally, it says that random sequence $x_0 x_1 \ldots$ has limit frequency $1/2$, i.e.,

$$\lim_{n\to\infty} \frac{x_0 + x_1 + \ldots + x_{n-1}}{n} = \frac{1}{2}.$$

However, the word "random" here is used only as a shortcut: the full meaning is that the set of all sequences that do not satisfy the Strong Law of Large Numbers (do not have limit frequency or have it different from $1/2$) is a null set.

In general, "$P(\omega)$ is true for random $\omega \in \Omega$" means that the set

$$\{\omega \mid P(\omega) \text{ is false}\}$$

is a null set.

Proof sketch: it is enough to show that for every $\delta > 0$ the set N_δ of sequences that have frequency greater than $1/2 + \delta$ for infinitely many prefixes, has measure 0. (After that we use that a countable union of null sets is a null set.) For each n consider the probability $p(n, \delta)$ of the event "random string of length n has more than $(1/2 + \delta)n$ ones". The crucial observation is that

$$\sum_n p(n, \delta) < \infty$$

for any $\varepsilon > 0$. (Actually, $p(n, \varepsilon)$ is exponentially decreasing when $n \to \infty$; proof uses Stirling's approximation for factorials.) If the series above has a finite sum, for every $\varepsilon > 0$ one can find an integer N such that

$$\sum_{n>N} p(n, \delta) < \varepsilon.$$

Consider all strings z of length greater than N that have frequency of ones greater than $1/2 + \delta$. The sum of $P(\Omega_z)$ is equal to $\sum_{n>N} p(n, \delta) < \varepsilon$, and N_ε is covered by family Ω_z. (End of proof sketch.)

28 Effectively null sets

The following notion was introduced by Per Martin-Löf. A set $X \subset \Omega$ is an *effectively null* set if there is an algorithm that gets a rational number $\varepsilon > 0$ as input and enumerates a set of strings $\{x_0, x_1, x_2, \ldots\}$ such that

(1) $X \subset \Omega_{x_0} \cup \Omega_{x_1} \cup \Omega_{x_2} \cup \ldots$;
(2) $\sum_i 2^{-|x_i|} < \varepsilon$.

The notion of effectively null set remains the same if we allow only ε of form $1/2^k$, or if we replace "$<$" by "$\leqslant$" in (2).

Any subset of an effectively null set is also an effectively null set (evident observation).

A singleton $\{\omega\}$ (containing some infinite sequence of zeros and ones) is a null set if ω is computable (or non-random, see below).

An union of two effectively null sets is an effectively null set. (Indeed, we can find enumerable coverings of size $\varepsilon/2$ for both and combine them.)

More general statement requires preliminary definition. By "covering algorithm" for an effectively null set we mean algorithm mentioned in the definition (that gets ε and generates a covering sequence of strings with sum of measures less than ε).

Lemma. Let $X_0, X_1, X_2, \ldots$ be a sequence of effectively null sets such that there exists an algorithm that for any input i produces (some) covering algorithm for X_i. Then $\cup X_i$ is an effectively null set.

Proof. To get an ε-covering for $\cup X_i$, we put together $(\varepsilon/2)$-covering for X_0, $(\varepsilon/4)$-covering for X_1, etc. To generate this combined covering, we use algorithm that produces covering for X_i from i. (End of proof.)

29 Maximal effectively null set

Up to now the theory of effectively null sets just repeats classical theory of null sets. The crucial difference is in the following theorem (proved by Martin-Löf):

Theorem 18 *There exists a maximal effectively null set, i.e., an effectively null set N such that $X \subset N$ for any effectively null set X.*

(Trivial) reformulation: the union of all effectively null sets is an effectively null set.

We cannot prove this theorem by applying Lemma above to all effectively null sets (there are uncountably many of them, since any subset of an effectively null set is an effectively null set).

But we don't need to consider all effectively null sets; it is enough to consider all covering algorithms. For a given algorithm (that gets positive rational number as input and generates binary strings) we cannot say (effectively) whether it is a covering algorithm or not. But we may artificially enforce some restrictions: if algorithm (for given $\varepsilon > 0$) generates strings $x_0, x_1, \ldots$, we can check whether $2^{-|x_0|} + \ldots + 2^{-|x_k|} < \varepsilon$ or not; if not, we delete x_k from generated sequence. Let us denote by A' the modified algorithm (if A was an original one). It is easy to see that

(1) if A was a covering algorithm for some effectively null set, then A' is equivalent to A (the condition that we enforce is never violated).

(2) For any A algorithm A' is (almost) a covering algorithm for some null set (the only difference is that the infinite sum $\sum 2^{-|x_i|}$ can be equal to ε even if all finite sums are strictly less than ε.

But this is not important: we can apply the same arguments (that were used to prove Lemma) to all algorithms $A'_0, A'_1, \ldots$ where $A_0, A_1, \ldots$ is a sequence of all algorithms (that get positive rational numbers as inputs and generate binary strings). (End of proof.)

Definition. A sequence ω of zeros and ones is called (Martin-Löf) *random* with respect to uniform Bernoulli measure if ω does not belong to maximal effectively null set.

(Reformulation: "...if ω does not belong to any effectively null set.")

Therefore, to prove that some sequence is non-random, it is enough to show that it belongs to some effectively null set.

Note also that a set X is an effectively null set if and only if all elements of X are non-random.

This sounds like a paradox for people familiar with classical measure theory. Indeed, we know that measure somehow reflects the "density" of set. Each point is a null set, but if we have too many points, we get a non-null set. Here (in Martin-Löf theory) if any element of a set forms an effectively null singleton (i.e., is non-random), then the whole set in an effectively null one.

Problems

1. Prove that if sequence $x_0x_1x_2\ldots$ of zeros and ones is (Martin-Löf) random with respect to uniform Bernoulli measure, then the sequence $000x_1x_2\ldots$ is also random. Moreover, adding any finite prefix to random sequence, we get a random sequence, and adding any finite prefix to non-random sequence, we get a non-random sequence.

2. Prove that any (finite) binary string appears infinitely many times in any random sequence.

3. Prove that any computable sequence is non-random. Give an example of a non-computable non-random sequence.

4. Prove that the set of all computable infinite sequences of zeros and ones is an effectively null set.

5*. Prove that if $x_0x_1\ldots$ is not random, then $n - K(x_0\ldots x_{n-1}|n) \to \infty$ as $n \to \infty$.

30 Gambling and selection rules

Richard von Mises suggested (around 1910) the following notion of a random sequence (he uses German word Kollektiv) as basis for probability theory. A sequence $x_0x_1x_2\ldots$ is called (Mises) random, if

(1) The limit frequency of 1's in $1/2$, i.e.,

$$\lim_{n\to\infty} \frac{x_0 + x_1 + \cdots + x_{n-1}}{n} = \frac{1}{2};$$

(2) the same is true for any infinite subsequence selected by an admissible selection rule.

Examples of admissible selection rules: (a) select terms with even indices; (b) select terms that follow zeros. The first rule gives $0100\ldots$ when applied to $\underline{0}0\underline{1}0\underline{0}1\underline{0}0\ldots$ (selected terms are underlined). The second rule gives $0110\ldots$ when applied to $0\underline{0}1\underline{0}1\underline{1}0\underline{0}\ldots$

Mises gave no exact definition of admissible selection rule (at that time the theory of algorithms was not developed). Later Church suggested the following formal definition of admissible selection rule.

An admissible selection rule is a total computable function S defined on finite strings that has values 1 ("select") and 0 ("do not select"). To apply S to a sequence $x_0x_1x_2\ldots$ we select all x_n such that $S(x_0x_1\ldots x_{n-1}) = 1$. Selected terms form a subsequence (finite or infinite). Therefore, each selection rule S determines a mapping $\sigma_S : \Omega \to \Sigma$, where Σ is the set of all finite and infinite sequences of zeros and ones.

For example, if $S(x) = 1$ for any string x, then σ_S is an identity mapping. Therefore, the first requirement in Mises approach follows from the second one, and we come to the following definition:

A sequence $x = x_0x_1x_2\ldots$ is *Mises–Church random*, if for any admissible selection rule S the sequence $\sigma_S(x)$ is either finite or has limit frequency $1/2$.

Church's definition of admissible selection rules has the following motivation. Imagine you come to a casino and watch the outcomes of coin tossing. Then you decide whether to participate in the next game or not, applying S to the sequence of observed outcomes.

31 Selection rules and Martin-Löf randomness

Theorem 19 *Applying admissible selection rule (according to Church definition) to Martin-Löf random sequence, we get either finite or Martin-Löf random sequence.*

Proof. Let S be a function that determines selection rule σ_S.

Let Σ_x be the set of all finite of infinite sequences that have prefix x (here x is a finite binary string).

Consider the set $A_x = \sigma_X^{-1}(\Sigma_x)$ of all (infinite) sequences ω such that selected subsequence starts with x. If $x = \Lambda$ (empty string), then $A_x = \Omega$.

Lemma. The set A_x has measure at most $2^{-|x|}$.

Proof. What is A_0? In other terms, what is the set of all sequences ω such that the selected subsequence (according to selection rule σ_S) starts with 0? Consider the set B of all strings z such that $S(z) = 1$ but $S(z') = 0$ for any prefix z' of string z. These strings are places where the first bet is made. Therefore,

$$A_0 = \cup\{\Omega_{z0} \mid z \in B\}$$

and

$$A_1 = \cup\{\Omega_{z1} \mid z \in B\}.$$

In particular, the sets A_0 and A_1 have the same measure and are disjoint, therefore

$$P(A_0) = P(A_1) \leqslant \frac{1}{2}.$$

From the probability theory viewpoint, $P(A_0)$ $[P(A_1)]$ is the probability of the event "the first selected term will be 0 [resp. 1]", and both events have the same probability (that does not exceed $1/2$) for (almost) evident reasons.

We can prove in the same way that A_{00} and A_{01} have the same measure. (See below for details.) Since they are disjoint subsets of A_0, both of them have measure at most $1/4$. The sets A_{10} and A_{11} also have equal measure and are subsets of A_1, therefore both have measure at most $1/4$, etc.

[Let us give an explicit description of A_{00}. Let B_0 be the set of all strings z such that

(1) $S(z) = 1$;

(2) there exists exactly one proper prefix z' of z such that $S(z') = 1$;

(3) $z'0$ is a prefix of z.

In other terms, B_0 corresponds to the positions where we are making our second bet while our first bet produces 0. Then

$$A_{00} = \cup\{\Omega_{z0} \mid z \in B_0\}$$

and

$$A_{01} = \cup\{\Omega_{z1} \mid z \in B_0\}.$$

Therefore A_{00} and A_{01} indeed have equal measures.] (Lemma in proved.)

It is also clear that A_x is the union of intervals Ω_y that can be effectively generated if x is known. (Here we use the computability of S.

Let $\sigma_S(\omega)$ be an infinite non-random sequence. Then $\{\omega\}$ is effectively null singleton. Therefore, for each ε one can effectively generate intervals $\Omega_{x_1}, \Omega_{x_2}, \ldots$ whose union covers $\sigma_S(\omega)$. The preimages $\sigma_S^{-1}(\Omega_{x_1}), \sigma_S^{-1}(\Omega_{x_2}), \ldots$ cover ω. Each of these preimages is an enumerable union of intervals, and if we combine all these intervals we get a covering for ω that has measure less than ε. Thus, ω is non-random (a contradiction).

Theorem is proved.

Theorem 20 *Any Martin-Löf random sequence has limit frequency* $1/2$.

Proof. By definition this means that the set $\neg$*SLLN* of all sequences that do not satisfy Strong Law of Large Numbers is an effectively null set. As we have mentioned, this is a null set and the proof relies on an upper bound for binomial coefficients. This upper bound is explicit, and the argument showing that the set $\neg$*SLLN* is a null set can be extended to show that $\neg$*SLLN* is an effectively null set. (End of proof.)

Combining these two results, we get the following

Theorem 21 *Any Martin-Löf random sequence is also Mises–Church random.*

Problems

1. The following selection rule is *not* admissible according to Mises definition: choose all terms x_{2n} such that $x_{2n+1} = 0$. Show that (nevertheless) it gives (Martin-Löf) random sequence if applied to a Martin-Löf random sequence.

2. Let $x_0x_1x_2\ldots$ be a Mises–Church random sequence. Let $a_N = |\{n < N \mid a_n = 0, a_{n+1} = 1\}|$. Prove that $a_N/N \to 1/4$ as $N \to \infty$.

32 Probabilistic machines

Consider a Turing machine that has access to source of random bits. It has some special states a, b, c with the following properties: after machine comes to a, it jumps to one of the states b and c with probability $1/2$ for each.

Or consider a program in some language that allows assignments

$$a := \mathit{random};$$

where *random* is a keyword and a is a variable that gets value 0 or 1 (with probability $1/2$; each new random bit is independent of others).

For a deterministic machine output is a function of input. Now it is not the case: for a given input machine can produce different outputs, and each output has some probability. In other terms, for any given input machine's output is a random variable.

Our goal is to find out what distribution this random variable may have. But let us consider a simpler question first. Let M be a machine that does not have input. (For example, M can be a Turing machine that is put to work on an empty tape, or a Pascal program that does not have `read` statements.) Now consider probability of the event "M terminates". What can be said about this number?

More formally, for each sequence $\omega \in \Omega$ we consider the behaviour of M if random bits are taken from ω. For a given ω the machine either terminates or not. Then p is the measure of the set T of all ω such that M terminates using ω. It is easy to see that T is measurable. Indeed, T is a union of T_n, where T_n is the set of all ω such that M stops after at most n steps using ω. Each T_n is a union of intervals Ω_t for some strings t of length at most n (machine can use at most n random bits if it runs in time n) and therefore is measurable.

A real number p is called *enumerable from below* or *semicomputable from below* if p is a limit of increasing computable sequence of rational numbers: $p = \lim p_i$, where $p_0 \leqslant p_1 \leqslant p_2 \leqslant \ldots$ and there is an algorithm that computes p_i given i.

Lemma. A real number p is enumerable from below if and only if the set $X_p = \{r \in \mathbb{Q} \mid r < p\}$ is enumerable.

Proof. (1) Let p be the limit of computable increasing sequence p_i. For any rational number r

$$r < p \Leftrightarrow \exists i\, r < p_i.$$

Let $r_0, r_1, \ldots$ be a computable sequence of rational numbers such that any rational number appears infinitely often in this sequence. The following algorithm enumerates X_p: at ith step, compare r_i and p_i; if $r_i < p_i$, output r_i.

(2) If X_p is enumerable, let $r_0, r_1, r_2, \ldots$ be its enumeration. Then $p_n = \max(r_0, r_1, \ldots, r_n)$ is an increasing computable sequence of rational numbers that converges to p. (End of proof.)

Theorem 22 *(a) Let M be a probabilistic machine without input. Then M's probability of termination is enumerable from below.*

(b) Let p be any real number in $[0,1]$ enumerable from below. Then there exists a probabilistic machine that terminates with probability p.

Proof. (a) Let M be any probabilistic machine. Let p_n be the probability that M terminates after at most n steps. The number p_n is a rational number with denominator 2^n that can be effectively computed for any given n. (Indeed, machine M can use at most n random bits during n steps. For each of 2^n binary strings we simulate behaviour of M and see for how many of them M terminates.) The sequence $p_0, p_1, p_2 \ldots$ is an increasing computable sequence of rational numbers that converges to p.

(b) Let p be any real number in $[0,1]$ enumerable from below. Let $p_0 \leqslant p_1 \leqslant p_2 \leqslant \ldots$ be an increasing computable sequence that converges to p. Consider the following probabilistic machine. It treats random bits $b_0, b_1, b_2 \ldots$ as binary digits of a real number

$$\beta = 0.b_0 b_1 b_2 \ldots$$

When i random bits are generated, we have lower and upper bounds for β that differ by 2^{-i}. If the upper bound β_i turns out to be less than p_i, machine terminates. It is easy to see that machine terminates for given $\beta = 0.b_0 b_1 \ldots$ if and only if $\beta < p$. Indeed, if upper bound for β is less than lower bound for p, then $\beta < p$. On the other hand, if $\beta < p$, then $\beta_i < p_i$ for some i (since $\beta_i \to \beta$ and $p_i \to p$ as $i \to \infty$). (End of proof.)

Now we consider probabilities of different outputs. Here we need the following definition: A sequence $p_0, p_1, p_2 \ldots$ of real numbers is *enumerable from below*, if there is a computable total function p of two variables (that range over natural numbers) with rational values (with special value $-\infty$ added) such that

$$p(i,0) \leqslant p(i,1) \leqslant p(i,2) \ldots$$

and

$$p(i,0), p(i,1), p(i,2), \ldots \to p_i$$

for any i.

Lemma. A sequence $p_0, p_1, p_2, \ldots$ of reals is enumerable from below if and only if the set of pairs

$$\{\langle i, r\rangle \mid r < p_i\}$$

is enumerable.

Proof. Let $p_0, p_1, \ldots$ be enumerable from below and $p_i = \lim_n p(i,n)$. Then

$$r < p_i \Leftrightarrow \exists n\,[r < p(i,n)]$$

and we can check $r < p(i,n)$ for all pairs $\langle i, r\rangle$ and for all n. If $r < p(i,n)$, pair $\langle i, r\rangle$ is included in the enumeration.

On the other hand, if the set of pairs is enumerable, for each n we let $p(i,n)$ be the maximum value of r for all pairs $\langle i, r\rangle$ (with given i) that appear during n steps of the enumeration process. (If there are no pairs, $p(i,n) = -\infty$.) Lemma is proved.

Theorem 23 (*a*) *Let M be a probabilistic machine without input that can produce natural numbers as outputs. Let p_i be the probability of the event "M terminates with output i". Then sequence $p_0, p_1, \ldots$ is enumerable from below and $\sum_i p_i \leqslant 1$.*

(*b*) *Let $p_0, p_1, p_2 \ldots$ be a sequence of non-negative real numbers that is enumerable from below, and $\sum_i p_i \leqslant 1$. Then there exists a probabilistic machine M that outputs i with probability (exactly) p_i.*

Proof. Part (a) is similar to the previous argument: let $p(i,n)$ be the probability that M terminates with output i after at most n steps. Than $p(i,0), p(i,1), \ldots$ is a computable sequence of increasing rational numbers that converges to p_i.

(b) is more complicated. Recall the proof of the previous theorem. There we had a "random real" β and "termination region" $[0,p)$ where p was the desired termination probability. (If β is in termination region, machine terminates.)

Now termination region is divided into parts. For each output value i there is a part of termination region that corresponds to i and has measure p_i. Machines terminates with output i if and only if β is inside ith part.

Let us consider first a special case when sequence p_i is a computable sequence of rational numbers, Then ith part is a segment of length p_i. These segments are allocated from left to right according to "requests" p_i. One can say that each number i comes with request p_i for space allocation, and this request is granted. Since we can compute the endpoints of all segments, and have lower and upper bound for β, we are able to detect the point when β will for sure be inside i-th part. (And if β is inside ith part, this will be detected at some step.)

In general case construction should be modified. Now each i come to space allocator many times with increasing requests $p(i,0), p(i,1), p(i,2)\ldots$ Each time the request is granted by allocating additional segment of length $p(i,n) - p(i,n-1)$. Note that ith part is not contiguous: it consists of infinitely many segments separated by other parts. But for now it is not important. Machine terminates with input i when current lower and upper bounds for β guarantee that β is inside ith part. The interior of ith part is a countable union of intervals, and if β is inside this open set, machine will terminate with output i. Therefore, termination probability is the measure of this set, i.e., equals $\lim_n p(i,n)$.

Theorem is proved.

Problems

1. Probabilistic machine without input terminates for all possible coin tosses (there is no sequence of coin tosses that leads to infinite computation). Prove that the computation time is bounded by some constant (and machine can produce only finite number of outputs).

2. Let p_i be the probability of termination with output i for some probabilistic machine and $\sum p_i = 1$. Prove that all p_i are computable, i.e., for any given i and for any rational $\varepsilon > 0$ we can find (algorithmically) an approximation to p_i with absolute error at most ε.

33 A priori probability

A sequence of real numbers $p_0, p_1, p_2, \ldots$ is called an *enumerable from below semimeasure* if there exists a probabilistic machine (without input) that produces i with probability p_i. (As we know, $p_0, p_1, \ldots$ is a enumerable from below semimeasure if and only if p_i is enumerable from below and $\sum p_i \leqslant 1$.)

The same definition can be used for real-valued functions on strings instead of natural numbers (probabilistic machines produce strings; the sum $\sum p(x)$ is taken over all strings x, etc.)

Theorem 24 *There exists a maximal enumerable from below semimeasure m (for any enumerable from below semimeasure m' there exists a constant c such that $m'(i) \leqslant cm(i)$ for all i).*

Proof. Let $M_0, M_1, \ldots$ be a sequence of all probabilistic machines without input. Let M be a machine that starts with choosing natural number i at random (so that any outcome has positive probability) and then emulates M_i. If p_i is the probability that i is chosen, m is the distribution on the outputs of M and m' is the distribution on the outputs of M_i, then $m(x) \geqslant p_i m'(x)$ for any x.

The maximal enumerable from below semimeasure is called *a priori probability*. This name can be explained as follows. Imagine that we have a black box that can be turned on and prints a natural number. We have no information about what is inside. Nevertheless we have an "a priori" upper bound for probability of the event "i appears" (up to a constant factor that depends on the box but not on i).

34 Prefix decompression

A priory probability is related to a special complexity measure called prefix complexity. The idea is that description is self-delimited; the decompression program had to decide for itself where to stop reading input. There are different versions of machines with self-delimiting input; we choose one that is technically convenient though may be not the most natural one.

A computable function whose inputs are binary strings is called a *prefix* function, if for any string x and its prefix y at least one of the values $f(x)$ and $f(y)$ is undefined. (So a prefix function cannot be defined both on a string and its prefix or continuation.)

Theorem 25 *There exists a prefix decompressor D that is optimal among prefix decompressors: for any computable prefix function D' there exists some constant c such that*

$$K_D(x) \leqslant K_{D'}(x) + c$$

for all x.

Proof. To prove similar theorem for plain Kolmogorov complexity we used

$$D(\overline{p}01y) = p(y)$$

where $\overline{p}$ is a program p with doubled bits and $p(y)$ stands for the output of program p with input y. This D is a prefix function if and only if all programs compute prefix functions. We cannot algorithmically distinguish between prefix and non-prefix programs (this is an undecidable problem). However, we may convert each program into a prefix one in such a way that prefix programs remain unchanged.

Let

$$D(\overline{p}01y) = [p](y)$$

where $[p](y)$ is computed as follows. We apply in parallel p to all inputs and get a sequence of pairs $\langle y_i, z_i\rangle$ such that $p(y_i) = z_i$. Select a "prefix" subsequence by deleting all $\langle y_i, z_i\rangle$ such that y_i is a prefix of y_j or y_j is a prefix of y_i for some $j < i$. This process does not depend on y. To compute $[p](y)$, wait until y appears in the selected subsequence, i.e. $y = y_i$ for a selected pair $\langle y_i, z_i\rangle$, and then output z_i.

The function $y \mapsto [p](y)$ is a prefix function for any p, and if program p computes a prefix function, then $[p](y) = p(y)$.

Therefore, D is an optimal prefix decompression algorithm. Theorem is proved.

Complexity with respect to an optimal prefix decompression algorithm is called *prefix complexity* and denoted by $KP(x)$ [LV use $K(x)$ while using $C(x)$ for plain Kolmogorov complexity.]

35 Prefix complexity and length

As we know, $K(x) \leqslant |x| + O(1)$ (consider identity mapping as decompression algorithm). But identity mapping is not a prefix one, so we cannot use this argument to show that $KP(x) \leqslant |x| + O(1)$, and in fact this is not true, as the following theorem shows.

Theorem 26

$$\sum_x 2^{-KP(x)} \leqslant 1$$

Proof. For any x let p_x be the shortest description for x (with respect to given prefix decompression algorithm). Then $|p_x| = KP(x)$ and all strings p_x are incompatible. (We say that p and q are compatible if p is a prefix of q or vice versa.) Therefore, intervals Ω_{p_x} are disjoint; they have measure $2^{-|p_x|} = 2^{-KP(x)}$, so the sum does not exceed 1. (End of proof.)

If $KP(x) \leqslant |x| + O(1)$ were true, then $\sum_x 2^{-|x|}$ would be finite, but it is not the case (for each natural number n the sum over strings of length n equals 1).

However, we can prove weaker lower bounds:

Theorem 27

$$KP(x) \leqslant 2|x| + O(1);$$
$$KP(x) \leqslant |x| + 2\log|x| + O(1);$$
$$KP(x) \leqslant |x| + \log|x| + 2\log\log|x| + O(1)$$
$$\textit{etc.}$$

Proof. The first bound is obtained if we use $D(\bar{x}01) = x$. (It is easy to check that D is prefix function.) The second one uses

$$D(\overline{\mathrm{bin}(|x|)}01x) = x$$

where $\mathrm{bin}(|x|)$ is the binary representation of the length of string x. Iterating this trick, we let

$$D(\overline{\mathrm{bin}(|\,\mathrm{bin}(|x|)\,|)}01\,\mathrm{bin}(|x|)x) = x$$

and get the third bound etc. (End of proof.)

Let us note that prefix complexity does not increase when we apply algorithmic transformation: $KP(A(x)) \leqslant KP(x) + O(1)$ for any algorithm A. Let us take optimal decompressor (for plain complexity) as A. We conclude that $KP(x)$ does not exceed $KP(p)$ where p is any description of x. Combining this with theorem above, we conclude that $KP(x) \leqslant 2K(x) + O(1)$, that $KP(x) \leqslant K(x) + 2\log K(x) + O(1)$, etc.

36 A priory probability and prefix complexity

We have now two measures for a string (or natural number) x. A priori probability $m(x)$ measures how probable is to see x as an output of a probabilistic machine. Prefix complexity measures how difficult is to specify a string in a self-delimiting way. It turns out that these two measures are closely related.

Theorem 28

$$KP(x) = -\log m(x) + O(1)$$

(Here $m(x)$ is a priori probability; log stands for binary logarithm.)

Proof. Function KP is enumerable from above; therefore, $x \mapsto 2^{-KP(x)}$ is enumerable from below. Also we know that $\sum_x 2^{-KP(x)} \leqslant 1$, therefore $2^{-KP(x)}$ is an enumerable from below semimeasure. Therefore, $2^{-KP(x)} \leqslant cm(x)$ and $KP(x) \geqslant -\log m(x) + O(1)$. To prove that $KP(x) \leqslant -\log m(x) + O(1)$, we need the following lemma about memory allocation.

Let the memory space be represented by $[0,1]$. Each memory request asks for segment of length $1, 1/2, 1/4, 1/8$, etc. that is properly aligned. Alignment means that for segment of length $1/2^k$ only 2^k positions are allowed ($[0, 2^{-k}], [2^{-k}, 2\cdot 2^{-k}]$, etc.). Allocated segments should be disjoint (common endpoints are allowed). Memory is never freed.

Lemma. For each computable sequence of requests 2^{-n_i} such that $\sum 2^{-n_i} \leqslant 1$ there is a computable sequence of allocations that grant all requests.

Proof. We keep a list of free space divided into segments of size 2^{-k}. Invariant relation: all segments are properly aligned and have different size. Initially there is one free segment of length 1. When a new request of length w comes, we pick up the smallest segment of length at least w. This strategy is sometimes called "best fit" strategy. (Note that if the free list contains only segments of length $w/2, w/4, \ldots,$

then the total free space is less than w, so it cannot happen by our assumption.) If smallest free segment of length at least w has length w, we simple allocate it (and delete from the free list). If it has length $w' > w$, then we divide w' into parts of size $w, w, 2w, 4w, \ldots, w'/4, w'/2$ and allocate the left w-segment putting all others in the free list, so the invariant is maintained. Lemma is proved.

Reformulation: ... there is a computable sequence of incompatible strings x_i such that $|x_i| = n_i$. (Indeed, an aligned segment of size 2^{-n} is I_x for some string x for length n.)

Corollary: ... $KP(i) \leqslant n_i$.

(Indeed, consider a decompressor that maps x_i to i. Since all x_i are pairwise incompatible, it is a prefix function.)

Now we return to the proof. Since m is enumerable from above, there exists a function $M : \langle x,k\rangle \mapsto M(x,k)$ of two arguments with rational values that is non-decreasing with respect to the second argument such that $\lim_k M(x,k) = m(x)$.

Let $M'(x,k)$ be the smallest number in the sequence $1, 1/2, 1/4, 1/8, \ldots, 0$ that is greater than or equal to $M(x,k)$. It is easy to see that $M'(x,k) \leqslant 2M(x,k)$ and that M' is monotone.

We call pair $\langle x,k\rangle$ "essential" if $k = 0$ or $M'(x,k) > M'(x,k-1)$. The sum of $M'(x,k)$ for all essential pairs with given x is at most twice bigger than its biggest term (because each term is at least twice bigger than preceding one), and its biggest term is at most twice bigger than $M(x,k)$ for some k. Since $M(x,k) \leqslant m(x)$ and $\sum m(x) \leqslant 1$, we conclude that sum of $M'(x,k)$ for all essential pairs $\langle x,k\rangle$ does not exceed 4.

Let $\langle x_i, k_i\rangle$ be a computable sequence of all essential pairs. (We enumerate all pairs and select essential ones.) Let n_i be an integer such that $2^{-n_i} = M'(x_i,k_i)/4$. Then $\sum 2^{-n_i} \leqslant 1$.

Therefore, $KP(i) \leqslant n_i$. Since x_i is obtained from i by an algorithm, we conclude that $KP(x_i) \leqslant n_i + O(1)$ for all i. For a given x one can find i such that $x_i = i$ and $2^{-n_i} \geqslant m_i/4$, i.e., $n_i \leqslant -\log m(x) + 2$, therefore $KP(x) \leqslant -\log m(x) + O(1)$.

Theorem is proved.

37 Prefix complexity of a pair

We can define $KP(x,y)$ as prefix complexity of some code $[x,y]$ of pair $\langle x,y\rangle$. Different computable encodings give complexities that differ at most by $O(1)$.

Theorem 29

$$KP(x,y) \leqslant KP(x) + KP(y) + O(1).$$

Note that now we don't need $O(\log n)$ term that was needed for plain complexity.

Let us give two proofs of this theorem using prefix functions and a priori probability.

(1) Let D be the optimal prefix decompressor used in the definition of KP. Consider a function D' such that

$$D'(pq) = [D(p), D(q)]$$

for all strings p and q such that $D(p)$ and $D(q)$ are defined. Let us prove that this definition makes sense, i.e., that it does not lead to conflicts. Conflict happens if $pq = p'q'$ and $D(p), D(q), D(p'), D(q')$ are defined. But then p and p' are prefixes of the same string and are compatible, so $D(p)$ and $D(p')$ cannot be defined at the same time unless $p = p'$ (which implies $q = q'$).

Let us check that D' is a prefix function. Indeed, if it is defined for pq and $p'q'$ and pq is a prefix of $p'q'$, then (as we have seen) p and p' are compatible and (since $D(p)$ and $D(p')$ are defined) $p = p'$. Then q is a prefix of q', so $D(q)$ and $D(q')$ cannot be defined at the same time.

D' is computable (for given x we try all decompositions $x = pq$ in parallel). So we have a prefix algorithm D' such that $K_D([x,y]) \leqslant KP(x) + KP(y)$ and $KP(x,y) \leqslant KP(x) + KP(y) + O(1)$. (End of the first proof.)

(2) In terms of a priori probability we have to prove that

$$m([x,y]) \geqslant \varepsilon m(x)m(y)$$

for some positive ε and all x and y. Consider the function m' determined by the equation

$$m'([x,y]) = m(x)m(y)$$

(m' is zero for inputs that do not encode pairs of strings). We have

$$\sum_z m'(z) = \sum_{x,y} m'([x,y]) = \sum_{x,y} m(x)m(y) = \sum_x m(x) \sum_y m(y) \leqslant 1 \cdot 1 = 1.$$

Function m' is enumerable from below, so m' is a semimeasure. Therefore, it is bounded by maximal semimeasure (up to a constant factor). (End of the second proof.)

Prefix complexity and randomness

Theorem 30 *A sequence $x_0x_1x_2\ldots$ is Martin-Löf random if and only if there exists some constant c such that*

$$KP(x_0x_1\ldots x_{n-1}) \geqslant n-c$$

for all n.

Proof. We have to prove that sequence $x_0x_1x_2\ldots$ is *not* random if and only if for any c there exists n such that

$$KP(x_0x_1\ldots x_{n-1}) < n-c.$$

Proof. (if) A string u is called c-defective if $KP(u) < |u| - c$. We have to prove that the set of all sequences that have c-defective prefix for any c, is an effectively null set. It is enough to prove that the set of all sequences that have c-defective prefix can be covered by intervals with total measure 2^{-c}.

Note that the set of all c-defective strings is enumerable (since KP is enumerable from above). It remains to show that the sum $\sum 2^{-|u|}$ over all c-defective u does not exceed 2^{-c}. Indeed, if u is c-defective, then by definition $2^{-u} \leqslant 2^{-c}2^{-KP(u)}$. On the other hand, the sum of $2^{-KP(u)}$ over all u (and therefore over defective u) does not exceed 1.

(only-if) Let N be the set of all non-random sequences. N is an effectively null set. For each integer c consider a sequence of intervals

$$\Omega_{u(c,0)}, \Omega_{u(c,1)}, \Omega_{u(c,2)}, \ldots$$

that cover N and have total measure at most 2^{-2c}. Definition of effectively null set guarantees that such a sequence exists (and its elements can be effectively generated for any c).

For each c,i consider the integer $n(c,i) = |u(c,i)| - c$. For a given c the sum $\sum_i 2^{-n(c,i)}$ does not exceed 2^{-c} (because the sum $\sum_i 2^{-|u(c,i)|}$ does not exceed 2^{-2c}. Therefore the sum $\sum_{c,i} 2^{-n(c,i)}$ over all c and i does not exceed 1.

Consider a semimeasure M such that $M(u(c,i)) = 2^{-n(c,i)}$. Correction: It may happen that $u(c,i)$ coincide for different pairs c,i. So the correct definition is

$$M(x) = \sum \{2^{-n(c,i)} \mid u(c,i) = x\}.$$

M is enumerable from below since u and n are computable functions. Therefore, if m is universal semimeasure, $m(x) \geqslant \varepsilon M(x)$, so $KP(x) \leqslant -\log M(x) + O(1)$, and $KP(u(c,i)) \leqslant n(c,i) + O(1) = |u(c,i)| - c + O(1)$.

If some sequence $x_0x_1x_2\ldots$ belongs to the set N of non-random sequences, then it has prefixes of form $u(c,i)$ for any c, and for these prefixes the difference between length and KP is not bounded.

(End of proof.)

38 Strong law of large numbers revisited

Let p, q be positive rational numbers such that $p+q=1$. Consider the following semimeasure: a string x of length n with k ones and l zeros has probability

$$\mu(x) = \frac{c}{n^2} p^k q^l$$

where constant c is chosen in such a way that $\sum_n c/n^2 \leqslant 1$. It is indeed a semimeasure (sum over all strings x is at most 1, because sum of $\mu(x)$ over all strings x of given length n is $1/n^2$; $p^k q^l$ is a probability to get string x if coin is biased and has probabilities p and q).

Therefore, we conclude that $\mu(x)$ is bounded by a priori probability (up to a constant) and we get an upper bound

$$KP(x) \leqslant 2\log n + k(-\log p) + l(-\log q) + O(1)$$

for fixed p and q and for any string x of length n that has k ones and l zeros. If $p = q = 1/2$, we get the bound $KP(x) \leqslant n + 2\log n + O(1)$ that we already know. The new bound is biased: If $p > 1/2$ and $q < 1/2$, then $-\log p < 1$ and $-\log q > 1$, so we count ones with less weight than zeros, and new bound can be better for strings that have many ones and few zeros.

Assume that $p > 1/2$ and the fraction of ones in x is greater that p. Then our bound implies

$$KP(x) \leqslant 2\log n + np(-\log p) + nq(-\log q) + O(1)$$

(more ones make our bound only tighter). It can be rewritten as

$$KP(x) \leqslant nH(p,q) + 2\log n + O(1)$$

where $H(p,q)$ is Shannon entropy for two-valued distribution with probabilities p and q:

$$H(p,q) = -p\log p - q\log q.$$

Since $p+q=1$, we have function of one variable:

$$H(p) = H(p, 1-p) = -p\log p - (1-p)\log(1-p).$$

This function has a maximum at $1/2$; it is easy to check using derivatives that $H(p) = 1$ when $p = 1/2$ and $H(p) < 1$ when $p \neq 1/2$.

Corollary. For any $p > 1/2$ there exist a constant $\alpha < 1$ and a constant c such that

$$KP(x) \leqslant \alpha n + 2\log n + c$$

for any string x where frequency of 1's is at least p.

Therefore, any infinite sequence of zeros and ones that has infinitely many prefixes with frequency of ones at least $p > 1/2$, is not Martin-Löf random. This gives us a proof of a constructive version of Strong Law of Large Numbers:

Theorem 31 *For any Martin-Löf random sequence $x_0x_1x_2\ldots$ of zeros and ones*

$$\lim_{n\to\infty} \frac{x_0 + x_1 + \ldots + x_{n-1}}{n} = \frac{1}{2}.$$

Problems

1. Let D be a prefix decompression algorithm. Give a direct construction of a probabilistic machine that outputs i with probability at least $2^{-K_D(i)}$.

2.* Prove that $KP(x) \leqslant K(x) + KP(K(x))$

3. Prove that there exists an infinite sequence $x_0x_1\ldots$ and a constant c such that

$$K(x_0x_1\ldots x_{n-1}) \geqslant n - 2\log n + c$$

for all n.

Final exam

Note that:

A. It is *not* needed to solve all problems (some of them are quite difficult) to get a maximal grade: solving half of them is a very good result.

B. You are allowed to look in the books, including Li–Vitanyi book (this is especially useful for some problems, e.g., 12). However, you should prove all results that you use.

1. Let k_n be average complexity of binary strings of length n:

$$k_n = \left[\sum_{|x|=n} K(x)\right] / 2^n.$$

Prove that $k_n = n + O(1)$ ($|k_n - n| < c$ for some c and all n).

2. Prove that for Martin-Löf random sequence $a_0a_1a_2a_3\ldots$ the set of all i such that $a_i = 1$ is not enumerable (there is no program that generates elements of this set).

3. (Continued) Prove the same result for Mises–Church random sequence.

4. String $x = yz$ of length $2n$ is incompressible: $K(x) \geqslant 2n$; strings y and z have length n. Prove that $K(y), K(z) \geqslant n - O(\log n)$.

5. (Continued) Is the reversed statement (if y and z are incompressible, then $K(yz) = 2n + O(\log n)$) true?

6. Prove that if $K(y|z) \geqslant n$ and $K(z|y) \geqslant n$ for strings y and z of length n, then $K(yz) \geqslant 2n - O(\log n)$.

7. Prove that if x and y are strings of length n and $K(xy) \geqslant 2n$, then the length of any common subsequence u of x and y does not exceed $0.99n$. (A string u is a subsequence of a string v if u can be obtained from v by deleting some terms. For example, 111 is a subsequence of 010101, but 1110 and 1111 are not.)

8. Let $a_0a_1a_2\ldots$ and $b_0b_1b_2\ldots$ be Martin-Löf random sequences and $c_0c_1c_2\ldots$ be a computable sequence. Can sequence $(a_0 \oplus b_0)(a_1 \oplus b_1)(a_2 \oplus b_2)\ldots$ be a non-random one? (Here $a \oplus b$ denotes $a + b$ mod 2.)

9. (Continued) The same question for $(a_0 \oplus c_0)(a_1 \oplus c_1)(a_2 \oplus c_2)\ldots$

10. True or false: $K(x,y) \leqslant KP(x) + K(y) + O(1)$?

11. Prove that for any c there exists x such that $KP(x) - K(x) > c$.

12. Let $m(x)$ be a priori probability of string x. Prove that binary representation of real number $\sum_x m(x)$ is a Martin-Löf random sequence.

13. Prove that $K(x) + K(x,y,z) \leqslant K(x,y) + K(x,z) + O(\log n)$ for strings x, y, z of length at most n.

14. (Continued) Prove the similar result for prefix complexity with $O(1)$ instead of $O(\log n)$.