Quantum Algorithms
Lecture 5
Probabilistic algorithms and the class BPP I

Zhejiang University

Definitions. Amplification of probability

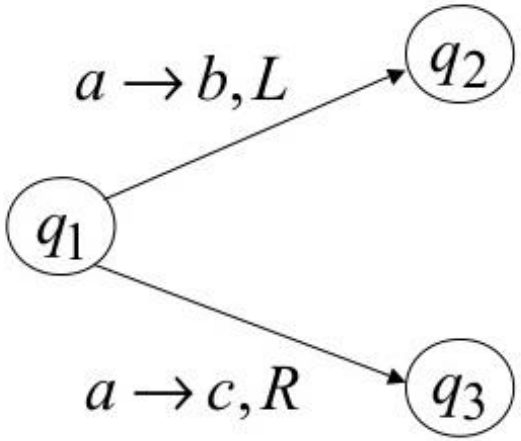
PTM

A probabilistic Turing machine (PTM) is somewhat similar to a nondeterministic one; the difference is that choice is produced by coin tossing, not by guessing.

For example:

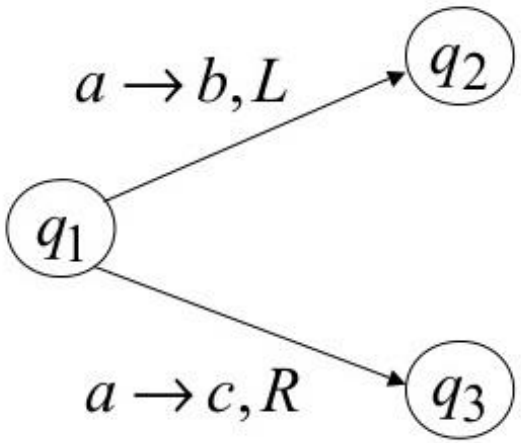
$\delta(q1, a) = (q2, b, \mathrm{L})$ with probability 1/2

$\delta(q1, a) = (q3, c, \mathrm{R})$ with probability 1/2



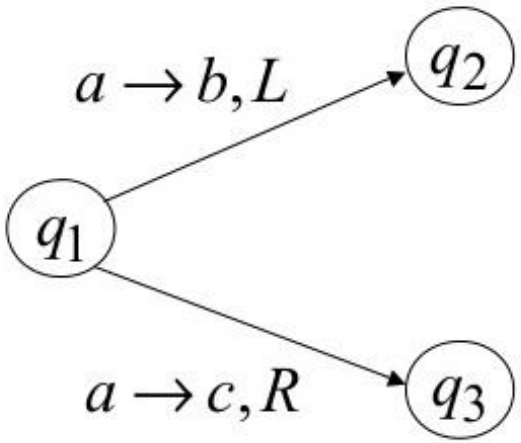
PTM

More formally, some (state,symbol) combinations have two associated actions, and the choice between them is made probabilistically. Each instance of this choice is controlled by a random bit. We assume that each random bit is 0 or 1 with probability 1/2 and that different random bits are independent.



PTM remark

In fact we can replace 1/2 by, say, 1/3 and get almost the same definition; the class BPP remains the same. However, if we replace 1/2 by some noncomputable real p, we get a rather strange notion which is better avoided.



Noncomputable real p

Let $x = x_1x_2x_3$ ... be an infinite binary sequence.

Binary probability value $p = 0.x_101x_201x_301$ ...

Recognition of any language (solving any predicate P(x):

Alphabet	Machine	Space	Time
unary	PTM	$O(n)$	$O(2^n)$
binary	PTM	$O(2^n)$	$O(2^{2^n})$
binary	1P4CA	$O(2^{2^n})$	$O(2^{2^n})$
binary	1P2CA	$O(2^{2^{2^n}})$	$O(2^{2^{2^n}})$

Published: Maksims Dimitrijevs and Abuzer Yakaryılmaz, Private-coin verification with magic coins. SOFSEM 2019: Current Trends in Theory and Practice of ComputerScience, Proceedings of Student Research Forum, pp. 1–12, 2019.

PTM results

For a given input string a PTM generates not a unique output string, but a probability distribution on the set of all strings (different values of the random bits lead to different computation outputs, and each possible output has a certain probability).

For example, for specific input x a PTM outputs 0 with probability 1/4, and 1 with probability 3/4.

BPP

Let ε be a constant such that $0 < \varepsilon < 1/2$. A predicate L belongs to the class BPP if there exist a PTM M and a polynomial $p(n)$ such that the machine M running on input string x always terminates after at most $p(|x|)$ steps, and

- $L(x) = 1 \Rightarrow M$ gives the answer “yes” with probability $\geq 1 - \varepsilon$;
- $L(x) = 0 \Rightarrow M$ gives the answer “yes” with probability $\leq \varepsilon$.

BPP probability

The admissible error probability ε can be, say, 0.49 or 10^{-10} — the class BPP will remain the same.

Assume that the PTM has probability of error at most $\varepsilon < 1/2$. Take k copies of this machine, run them all for the same input (using independent random bits) and pick the most frequent outcome.

Amplification of probability

To reduce the probability of error, we can repeat calculation and choose the majority of outputs.

Number of repetitions $= \theta(\log \frac{1}{\varepsilon})$,

where ε is error probability.

For error probability 2^{-q} num. of repetitions is $\sim q$

Example of outputs (run algorithm for 5 times): 1,0,1,0,1, then we see, that number 1 is more frequent (3 times out of 5), therefore our answer is 1.

Amplification of probability

Assume that error probability is ε. We run algorithm k times and choose the most frequent outcome as our result.

Probability of error is equal to the probability, that out of k runs at least $k/2$ are giving wrong result.

$$p_{error} \leq \sum_{S \subseteq \{1,\ldots,k\}, |S| \leq \frac{k}{2}} (1-\varepsilon)^{|S|} \varepsilon^{k-|S|} =$$

$$= ((1-\varepsilon)\varepsilon)^{k/2} \sum_{S \subseteq \{1,\ldots,k\}, |S| \leq \frac{k}{2}} \left(\frac{\varepsilon}{1-\varepsilon}\right)^{\frac{k}{2}-|S|}$$

Amplification of probability

Probability of error is equal to the probability, that out of k runs at least $k/2$ are giving wrong result.

$$p_{error} \leq \left((1-\varepsilon)\varepsilon\right)^{\frac{k}{2}} \sum_{S\subseteq\{1,\ldots,k\},|S|\leq\frac{k}{2}} \left(\frac{\varepsilon}{1-\varepsilon}\right)^{\frac{k}{2}-|S|} <$$

$$< \left(\sqrt{(1-\varepsilon)\varepsilon}\right)^{k} 2^{k} = \lambda^{k}, \text{ where } \lambda = 2\sqrt{\varepsilon(1-\varepsilon)} < 1$$

We need to set $k = \theta(\log\frac{1}{\varepsilon'})$ to have error probability ε'.

Even if we require that $\varepsilon' = \exp(-p(n))$ for an arbitrary polynomial p, the composite TM still runs in polynomial time.

Amplification of probability

If k is big enough, the effective error probability will be as small as we wish. This is called amplification of probability.

In fact, probabilistic computation can be as reliable as deterministic one.

Probability of error in case of deterministic computation appears because of the possibility of hardware to fail, which makes errors probable.

BPP - equivalent definition

A predicate L belongs to BPP if there exist a polynomial p and a predicate R, decidable in polynomial time, such that

- $L(x) = 1 \Rightarrow$ the fraction of strings r of length $p(|x|)$ satisfying $R(x, r)$ is greater than $1 - \varepsilon$;
- $L(x) = 0 \Rightarrow$ the fraction of strings r of length $p(|x|)$ satisfying $R(x, r)$ is less than ε.

Here strings r denote strings of random bits – e.g., probabilistic choices of PTM. This reminds us proof/solution like in NP definition.

Equivalence of BPP definitions

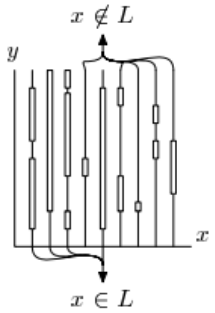
Let $R(x, r)$ be the following predicate: "M says 'yes' for the input x using $r_1, \ldots, r_{p_n}$ as the random bits" (we assume that a coin is tossed at each step of M). It is easy to see that the requirements of the second definition are satisfied.

Equivalence of BPP definitions

Assume that p and R are given. Consider a PTM that (for input x) randomly chooses a string r of length $p(|x|)$, making $p(|x|)$ coin tosses, and then computes $R(x,r)$. This machine satisfies definition of BPP with PTM (with a different polynomial p' instead of p).

BPP illustration

We represent a pair (x, y) of strings as a point and draw the set $S = \{(x, y)\colon (|y| = p(|x|)) \wedge R(x, y)\}$. For each x we consider the x-section of S defined as $S_x = \{y\colon (x, y) \in S\}$. The set S is rather special in the sense that, for any x, either S_x is large (contains at least $1 - \varepsilon$ fraction of all strings of length $p(|x|)$) or is small (contains at most ε fraction of them). Therefore, all values of x are divided into two categories: for one of them $L(x)$ is true and for the other $L(x)$ is false.



BPP illustration

All values of x are divided into two categories: for one of them $L(x)$ is true and for the other $L(x)$ is false.

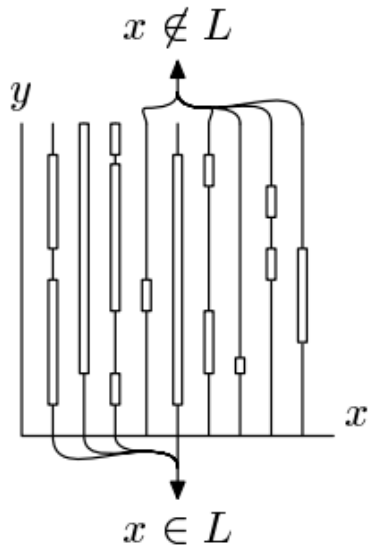


Fig. 4.1. The characteristic set of the predicate $R(x, y)$. Vertical lines represent the sets S_x.

PTMs are real

Probabilistic Turing machines (unlike nondeterministic ones, which depend on almighty Merlin for guessing the computational path) can be considered as real computing devices. Physical processes like the Nyquist noise or radioactive decay are believed to provide sources of random bits; in the latter case it is guaranteed by the very nature of quantum mechanics.

Random information can bring additional source of computational power.

Primality testing

Primality problem

A classic example of a BPP problem is checking whether a given integer q (represented by $n = \log_2 q$ bits) is prime or not.

We will describe a probabilistic primality test, called Miller–Rabin test.

Main idea

Fermat test

Based on Fermat's little theorem, which says that if q is prime, then $a^{q-1} = 1\,(mod\,q)$ for $a \in \{1, \ldots, q-1\}$.

Example: $q = 5$

$1^4 \equiv 1(mod\,5)$

$2^4 = 16 \equiv 1(mod\,5)$

$3^4 = 81 \equiv 1(mod\,5)$

$4^4 = 256 \equiv 1(mod\,5)$

Notation simplification

We may regard a as a $(mod\, q)$-residue and simply write $a^{q-1} = 1$, assuming that arithmetic operations are performed with residues rather than integers.

Fermat test

We pick a random a and check whether $a^{q-1} = 1$.

If this is true, then q may be a prime; but if this is false, then q is not a prime.

Such a can be called a witness saying that q is composite.

Fermat test

This kind of evidence is indirect (it does not give us any factor of q) but usually easy to find: it often suffices to check $a = 2$. But we will do a better job if we sample a from the uniform distribution over the set $\{1, \ldots, q-1\}$ (i.e., each element of this set is taken with probability $1/(q-1)$).

Fermat test

Suppose q is composite. We want to know if the test actually shows that with nonnegligible probability.

1) $\gcd(a, q) = d \neq 1$. Then $a^{q-1} \equiv 0 \neq 1 (mod\ d)$, therefore $a^{q-1} \neq 1 (mod\ q)$. The test detects that q is composite. Unfortunately, the probability to get such an a is usually small.

Fermat test

Suppose q is composite. We want to know if the test actually shows that with nonnegligible probability.

2) $\gcd(a, q) = 1$, i.e. $a \in (Z/qZ)^*$ (where $(Z/qZ)^*$ denotes **the group of invertible (mod q)-residues**). This is the typical case; let us consider it more closely.

Ring

A ring is a set R equipped with two binary operations, "+" and "·" and two special elements, 0 and 1, so that the following relations hold:

$(a+b)+c=a+(b+c)$, $a+b=b+a$, $a+0=a$,

$(ab)c=a(bc)$, $1\cdot a=a\cdot 1=a$,

$(a+b)c=ac+bc$, $c(a+b)=ca+cb$.

For any $a\in R$ there exists an element v such that $a+v=0$.

Commutative ring

If, in addition, $ab = ba$ for any a and b, then R is called a commutative ring.

Z/qZ

The ring of residues modulo q.
We can simplify the notation, for example:
$Z/7Z$ = {0,1,2,3,4,5,6}

Zero divisors

For integers, $xy = 0$ implies that $x = 0$ or $y = 0$. This is not true for all residue rings (specifically, this is false in the case where q is a composite number). Example: $2 \cdot 5 \equiv 0(mod\ 10)$, although both 2 and 5 represent nonzero elements of $Z/10Z$. We say that an element x of a ring R is a zero divisor if $\exists y \neq 0(xy = 0)$. For example 0, 2, 3, 4, 6, 8, 9, 10 are zero divisors in $Z/12Z$, whereas 1, 5, 7, 11 are not.

Invertible elements

An element $x \in R$ is called invertible if there exists y such that $xy = 1$; in this case we write $y = x^{-1}$. For example, $7 = 4^{-1}$ in $Z/9Z$, since $4 \cdot 7 \equiv 1\,(mod\,9)$. It is obvious that if a and b are invertible, then ab is also invertible, and that $(ab)^{-1} = a^{-1}b^{-1}$. Therefore, invertible elements form an Abelian group with respect to multiplication, which is denoted by R^*. For example, $Z^* = \{1, -1\}$ and $(Z/12Z)^* = \{1,5,7,11\}$. In the latter case, invertible elements are exactly the elements which are not zero divisors.

Abelian group

A set A, together with an operation $\cdot$ that combines any two elements a and b of A to form another element of A, denoted $a \cdot b$. Abelian group axioms:

- For all a, b in A, the result of the operation $a \cdot b$ is also in A.
- For all a, b, and c in A, the equation $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ holds.
- There exists an element e in A, such that for all elements a in A, the equation $e \cdot a = a \cdot e = a$ holds.
- For each a in A there exists an element b in A such that $a \cdot b = b \cdot a = e$, where e is the identity element.
- For all a, b in A, $a \cdot b = b \cdot a$ (commutativity).

Subgroup

G={0,2,4,6,1,3,5,7}

This group has two nontrivial subgroups: J={0,4} and H={0,2,4,6}, where J is also a subgroup of H.

+	0	2	4	6	1	3	5	7
0	0	2	4	6	1	3	5	7
2	2	4	6	0	3	5	7	1
4	4	6	0	2	5	7	1	3
6	6	0	2	4	7	1	3	5
1	1	3	5	7	2	4	6	0
3	3	5	7	1	4	6	0	2
5	5	7	1	3	6	0	2	4
7	7	1	3	5	0	2	4	6

Abelian group examples

Every cyclic group G is abelian, because if x, y are in G, then $x\,y = a^m a^n = a^{m+n} = a^n a^m = y\,x$. Thus the integers, Z, form an abelian group under addition, as do the integers modulo n, Z/nZ.

Every ring is an abelian group with respect to its addition operation.

Every subgroup of an abelian group is abelian.

Lagrange's theorem

Lagrange's theorem states that for any finite group G, the order of every subgroup of G divides the order of G.

Order of G - number of elements in G.

Example: G={0,2,4,6,1,3,5,7}, $|G| = 8$;

H={0,2,4,6}, $|H| = 4$;

J={0,4}, $|J| = 2$.

As we can see, 2 and 4 divide 8.

+	0	2	4	6	1	3	5	7
0	0	2	4	6	1	3	5	7
2	2	4	6	0	3	5	7	1
4	4	6	0	2	5	7	1	3
6	6	0	2	4	7	1	3	5
1	1	3	5	7	2	4	6	0
3	3	5	7	1	4	6	0	2
5	5	7	1	3	6	0	2	4
7	7	1	3	5	0	2	4	6

Lemma

If $a^{q-1} \neq 1$ for some element $a \in (Z/qZ)^*$, then the Fermat test detects the compositeness of q with probability $\geq 1/2$.

Proof of Lemma

Let $G = (Z/qZ)^*$. For any integer m define the following set:

$G_{(m)} = \{x \in G : x^m = 1\}$.

This is a subgroup in G (due to the identity $a^m b^m = (ab)^m$ for elements of an Abelian group).

Proof of Lemma

If $a^{q-1} \neq 1$ for some a, then $a \notin G_{(q-1)}$, therefore $G_{(q-1)} \neq G$. By Lagrange's theorem, the ratio $|G|/|G_{(m)}|$ is an integer, hence $|G|/|G_{(m)})| \geq 2$. It follows that $a^{q-1} \neq 1$ for at least half of $a \in (Z/qZ)^*$. And, as we already know, $a^{q-1} \neq 1$ for all $a \notin (Z/qZ)^*$.

Carmichael numbers

Is it actually possible that q is composite but $a^{q-1} = 1$ for all invertible residues a? Such numbers q are rare, but they exist (they are called Carmichael numbers). Example: $q = 561 = 3 \cdot 11 \cdot 17$. Note that the numbers $3 - 1$, $11 - 1$ and $17 - 1$ divide $q - 1$. Therefore $a^{q-1} = 1$ for any $a \in (Z/qZ)^* \cong Z_{3-1} \times Z_{11-1} \times Z_{17-1}$.

Another primality test

The Fermat test alone is not sufficient to detect a composite number. The Miller–Rabin test uses yet another type of witnesses for the compositeness: if $b^2 \equiv 1 (mod\, q)$, and $b \neq \pm 1\, (mod\, q)$ for some b, then q is composite. Indeed, in this case $b^2 - 1 = (b-1)(b+1)$ is a multiple of q but $b-1$ and $b+1$ are not, therefore q has nontrivial factors in common with both $b-1$ and $b+1$.

Miller-Rabin test example

- $n = 5*7*11 = 385$

  $n - 1 = 384 = 2^7*3$

  $k = 7, m = 3$

  $a = 9$

  $b_0 = 9^3 = 344 \pmod{385}$

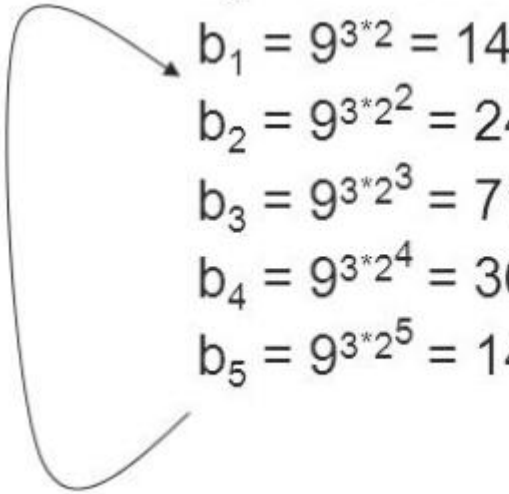
  $b_1 = 9^{3*2} = 141 \pmod{385}$

  $b_2 = 9^{3*2^2} = 246 \pmod{385}$

  $b_3 = 9^{3*2^3} = 71 \pmod{385}$

  $b_4 = 9^{3*2^4} = 36 \pmod{385}$

  $b_5 = 9^{3*2^5} = 141 \pmod{385}$



- $n = 3*11*17 = 561$

  $n - 1 = 560 = 2^4*35$

  $k = 4, m = 35$

  $a = 2$

  $b_0 = 2^{35} = 263 \pmod{561}$

  $b_1 = 2^{35*2} = 166 \pmod{561}$

  $b_2 = 2^{35*2^2} = 67 \pmod{561}$

  $b_3 = 2^{35*2^3} = 1 \pmod{561}$

  Proof of compositeness!

Required subroutines and their complexity

Basic operation complexity

Addition (or subtraction) of n-bit integers is done by an $O(n)$-size circuit; multiplication and division are performed by $O(n^2)$ -size circuits. These estimates refer to the standard algorithms learned in school, though they are not the most efficient for large integers.

Basic operation complexity

If only the size of the circuit is important, the standard addition algorithm is optimal, but the ones for the multiplication and division are not. For example, an $O(n\,logn\,loglogn)$-size circuit for the multiplication exists

Algorithm for GCD

Euclid's algorithm for $\gcd(a, b)$ has complexity $O(n^3)$, but there is also a so-called "binary" gcd algorithm of complexity $O(n^2)$. It does not solve the equation $xa + yb = \gcd(a, b)$, though.

See pages 247-248 of the book about Euclid's algorithm.

Modular arithmetic

It is clear that the addition of (mod q)-residues is done by a circuit of size $O(n)$, whereas modular multiplication can be reduced to integer multiplication and division; therefore it is performed by a circuit of size $O(n^2)$ (by the standard technique).

Modular arithmetic

To invert a residue $a \in (Z/qZ)^*$, we need to find an integer x such that $xa \equiv 1\,(mod\ q)$, i.e., $xa + yq = 1$. This is done by extended Euclid's algorithm, which has complexity $O(n^3)$.

Computing exponent in poly-time

It is also possible to compute $(a^m \, mod \, q)$ by a polynomial time algorithm. (Note that we speak about an algorithm that is polynomial in the length n of its input (a, m, q); therefore, performing m multiplications is out of the question. Note also that the size of the integer a^m is exponential.)

Computing exponent in poly-time

We can compute $(a^{2^k} \, mod \, q)$ for $k = 1, 2, \ldots, \log_2 m$ by repeated squaring, applying the $(mod \, q)$ reduction at each step. Then we multiply some of the results in such a way that the powers, i.e., the numbers 2^k, add to m (using the binary representation of m). This takes $O(\log m) = O(n)$ modular multiplications, which translates to circuit size $O(n^3)$.

Computing exponent in poly-time

It is also called exponentiation by squaring.

Suppose we want to calculate 2^{13}. In binary, $13 = 1101$.

1) we start with $2^0 = 1$

2) value of 1st bit is 1, so we do $2^0 * 2^0 * 2 = 2^1$

3) value of 2nd bit is 1, so we do $2^1 * 2^1 * 2 = 2^3$

4) value of 3rd bit is 0, so we do $2^3 * 2^3 = 2^6$

5) value of 4th bit is 1, so we do $2^6 * 2^6 * 2 = 2^{13}$

As we see, we square previous result in each step and additionally multiply by 2 when the bit value is equal to 1.

GCD properties

1. gcd(a, b) = gcd(b, a)
2. gcd(a, b) = gcd(-a, b) = gcd(a, -b) = gcd(-a, -b)
3. gcd(a, b) = gcd(b, remainder of (a/b))
4. gcd(a, 0) = |a|

Properties 3 and 4 are important for Euclid's algorithm.

Euclid's algorithm

Example

Calculate gcd(36,10)

gcd(36,10) =
gcd(10,6) =
gcd(6,4) =
gcd(4,2) =
gcd(2,0) = 2

a	b	q	r
36	10	3	6
10	6	1	4
6	4		

gcd(a,b) = gcd(b, r)

Extended Euclid's algorithm

Input: $a, b \in R$.

Output: $g \in R$ a gcd of a and b together with $s, t \in R$ such that $g = s\,a + t\,b$.

$r_0 := a$; $s_0 := 1$; $t_0 := 0$
$r_1 := b$; $s_1 := 0$; $t_1 := 1$
$i := 2$
while $r_{i-1} \neq 0$ **repeat**
  $q_i := r_{i-2}$ quo r_{i-1}
  $r_i := r_{i-2}$ rem r_{i-1}
  $s_i := s_{i-2} - q_i\, s_{i-1}$
  $t_i := t_{i-2} - q_i\, t_{i-1}$
  $i := i + 1$
return$(r_{i-2}, s_{i-2}, t_{i-2})$

Extended Euclid's algorithm

$\gcd(a, b) = d; ma + nb = d;$
$-9 \times 240 + 47 \times 46 = 2$
23 and -120 are quotients (up to a sign): $23 \times 2 = 46, |-120| \times 2 = 120$

index i	quotient q_{i-1}	Remainder r_i	s_i	t_i
0		240	1	0
1		46	0	1
2	240 ÷ 46 = 5	240 − 5 × 46 = 10	1 − 5 × 0 = 1	0 − 5 × 1 = −5
3	46 ÷ 10 = 4	46 − 4 × 10 = 6	0 − 4 × 1 = −4	1 − 4 × −5 = 21
4	10 ÷ 6 = 1	10 − 1 × 6 = 4	1 − 1 × −4 = 5	−5 − 1 × 21 = −26
5	6 ÷ 4 = 1	6 − 1 × 4 = 2	−4 − 1 × 5 = −9	21 − 1 × −26 = 47
6	4 ÷ 2 = 2	4 − 2 × 2 = 0	5 − 2 × −9 = 23	−26 − 2 × 47 = −120

Thank you for your attention!