Quantum Algorithms
Lecture 22
Quantum algorithms for Abelian groups I

Zhejiang University

Introduction

BQP $\supset$ BPP evidences

At present time, there is no proof that quantum computation is super-polynomially faster than classical probabilistic computation. But there are several pieces of indirect evidence in favor of such an assertion. The first of these is an example of a problem with oracle (so the algorithm does queries to an external device, like in Grover's Search) for which there exists a polynomial quantum algorithm, while any classical probabilistic algorithm is exponential.

BQP $\supset$ BPP evidences

The hidden subgroup problem for $(\mathbb{Z}_2)^k$.

The famous factoring algorithm by P. Shor is based on a similar idea.

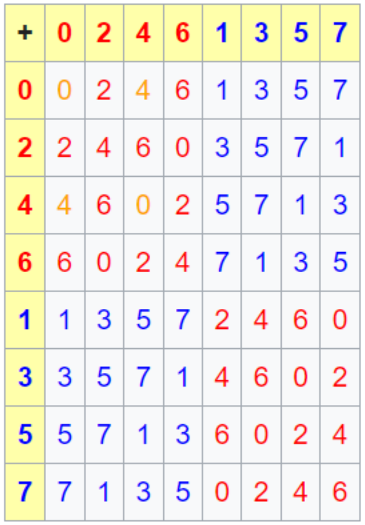
Then we will solve the hidden subgroup problem for the group $\mathbb{Z}^k$, which generalizes both results.

The hidden subgroup problem

Let G be a group with a specified representation of its elements by binary words. There is a device (an oracle) that computes some function $f\colon G \to B^n$ with the following property:

$$f(x) = f(y) \iff x - y \in D$$

where $D \subseteq G$ is an initially unknown subgroup. It is required to find that subgroup.



+	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

The HSP example

$f(x) = f(y) \iff x - y \in D$

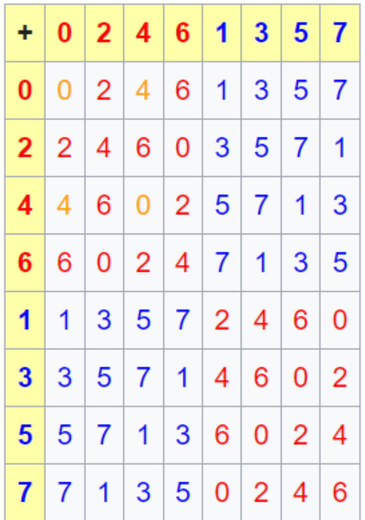
$G = \{0,1,2,3,4,5,6,7\}$

$D = \{0,4\}$

$f(0) = f(4), f(1) = f(5), f(2) = f(6), f(3) = f(7)$. $f(x)$ can have some values, e.g.:

$f(0) = 0,\ f(1) = 1,\ f(2) = 2, f(3) = 3;$ to fulfill property on the top of the slide all these 4 values should be different.

Query – we ask $f(x)$ from oracle by providing to it value x.



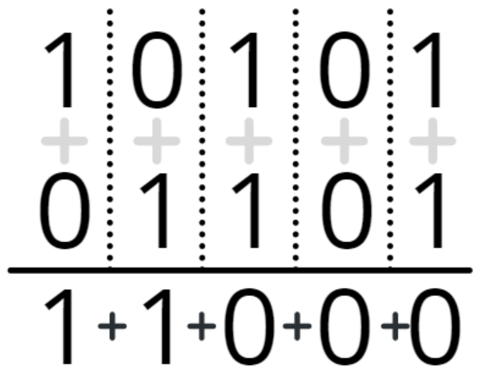
+	0	2	4	6	1	3	5	7
0	0	2	4	6	1	3	5	7
2	2	4	6	0	3	5	7	1
4	4	6	0	2	5	7	1	3
6	6	0	2	4	7	1	3	5
1	1	3	5	7	2	4	6	0
3	3	5	7	1	4	6	0	2
5	5	7	1	3	6	0	2	4
7	7	1	3	5	0	2	4	6

The problem of hidden subgroup in $(\mathbb{Z}_2)^k$; Simon's algorithm

HSP in $(\mathbb{Z}_2)^k$

We consider the problem formulated above for the group $G = (\mathbb{Z}_2)^k$. The elements of this group can be represented by length k words of zeros and ones; the group operation is bitwise addition modulo 2. We may regard G as the k-dimensional linear space over the field F_2. Any subgroup of G is a linear subspace, so it can be represented by a basis.

Example of elements and operation: Here 10101 and 01101 are elements of the group $G = (\mathbb{Z}_2)^5$



Classical limitations

Let $n \geq k$. For any classical probabilistic algorithm making no more than $2^{k/2}$ queries to the oracle, there exist a subgroup $D \subseteq (\mathbb{Z}_2)^k$ and a corresponding function f: $(\mathbb{Z}_2)^k \rightarrow B^n$ for which the algorithm is wrong with probability $> 1/3$.

So we have 2^k different x values, and if we ask (check) at most $\sqrt{2^k}$ $f(x)$ values, then error probability $> 1/3$ for some subgroups D.

Classical limitations - proof

For the same subgroup D there exist several different oracles f. We assume that one of them is chosen randomly and uniformly. (If the algorithm is wrong with probability $> 1/3$ for the randomized oracle, then it will be wrong with probability $> 1/3$ for some particular oracle.)

Why different oracles:

For $G = \{0,1,2,3,4,5,6,7\}$ $D = \{0,4\}$ and $f(0) = f(4), f(1) = f(5), f(2) = f(6), f(3) = f(7)$. $f(x)$ can have different valid oracles, e.g.:

$f(0) = 0$, $f(1) = 1$, $f(2) = 2, f(3) = 3$ OR

$f(0) = 1$, $f(1) = 2$, $f(2) = 3, f(3) = 0$.

Classical limitations - proof

The randomized oracle works as follows. If the present query is x_j, and $x_j - x_s \in D$ for some $s < j$, the answer y_j coincides with the answer y_s that was given before. Otherwise, y_j is uniformly distributed over the set $B^n \backslash \{y_1, \dots, y_{j-1}\}$. The randomized oracle is not an oracle in the proper sense, meaning that its answer may depend on the previous queries rather than only on the current one.

So Oracle should return $y_j = y_s$ if $x_j - x_s \in D$ or arbitrary number that was not present before if for no previous element x_s: $x_j - x_s \in D$.

Classical limitations - proof

In this manner, the randomized oracle is equivalent to a device with memory, which, being asked the question x_j, responds with the smallest number $s_j \leq j$ such that $x_j - x_{s_j} \in D$. Indeed, if we have a classical probabilistic machine making queries to the randomized oracle, we can adapt it for the use with the device just described. For this, the machine should be altered in such a way that it will transform each answer s_j to y_j and proceed as before. (That is, $y_j = y_s$ if $s_j < j$, or y_j is uniformly distributed over $B^n \backslash \{y_1, \dots, y_{j-1}\}$ if $s_j = j$.)

j is the number of query.

Classical limitations - proof

Let the total number of queries be $l \leq 2^{k/2}$. Without loss of generality all queries can be assumed different. In the case $D = \{0\}$, all answers are also different, i.e., $s_j = j$ for all j. Now we consider the case $D = \{0, z\}$, where z is chosen randomly, with equal probabilities, from the set of all nonzero elements of the group $(\mathbb{Z}_2)^k$.

Classical limitations - proof

Then, regardless of the algorithm that is used, $z \notin \{x_j - x_1, \ldots, x_j - x_{j-1}\}$ with probability $\geq 1 - (j-1)/(2^k - 1)$.

This condition for z implies that $s_j = j$. This is true for all $j = 1, \ldots, l$ with probability $\geq 1 - l(l-1)/(2(2^{k-1} - 1)) > 1/2$.

We can fix z in such a way that the probability of obtaining the answers $s_j = j$ (for all j) will still be greater than $1/2$. Let us see what a classical machine would do in such a circumstance.

Classical limitations - proof

If it gives the answer “$D = \{0\}$” with probability $\geq 2/3$, we set $D = \{0, z\}$, and then the resulting answer will be wrong with probability $> (2/3) \cdot (1/2) = 1/3$.

If, however, the probability of the answer “$D = \{0\}$” is smaller than $2/3$, we set $D = \{0\}$ and in such case probability of error will be $> 1/3$.

Remark: proof mentioned random parameter r just once, but it was not used. We may assume that it means random oracle – and we pick according z to have the desired probability.

Quantum oracle

Let us define a quantum analog of the oracle f. The corresponding quantum oracle is a unitary operator

$$U\colon |x, y\rangle \to |x, y \oplus f(x)\rangle$$

($\oplus$ denotes bitwise addition). We note that the quantum oracle allows linear combinations of the various queries; therefore it is possible to use it more efficiently than the classical oracle. For example, by doing queries in superposition.

Finding E^*

E = G/D denotes the quotient group.

Example:

If G={0,1,2,3,4,5} and D={0,3}, then E = G/D = {a+N: a∈G} = {{0,3},{1,4},{2,5}}={0+N,1+N,2+N}.

E^* the group of characters on E. (By definition, a character is a homomorphism E→U(1).)

The group U(1) corresponds to the group, consisting of all complex numbers with absolute value 1 under multiplication.

Finding E^*

E^*={$h \in (\mathbb{Z}_2)^k$ such that $h \cdot z = 0$ for all $z \in D$}
$h \cdot z = \sum_i h_i * z_i \ (mod\ 2)$,

So we have h and z represented by bits, $h_i * z_i = 1$ only if $h_i = z_i = 1$, and since we take modulo 2, $h \cdot z = 0$ only if $h_i = z_i = 1$ for even number of bits.

The character corresponding to h has the form $z \to (-1)^{h \cdot z}$. If $h \cdot z = 0$, then $z \to 1$.

Finding E^*

Let us show how one can generate a random element $h \in E^*$ using the operator U. After generating sufficiently many random elements, we will find the group E^*, hence the desired subgroup D.

Simon's algorithm

We start with two quantum registers:

$$|0^k\rangle \otimes |0^n\rangle$$

After that we apply Hadamard gate to each qubit of the first register (so we obtain an equal superposition):

$$|\xi\rangle = 2^{-k/2} \sum_{x \in G} |x\rangle = H^{\otimes k}|0^k\rangle$$

And we apply U: $|x, y\rangle \rightarrow |x, y \oplus f(x)\rangle$. Looks like imprecision in a book - U is applied to the combined system, so we have x and y.

Simon’s algorithm

Then we discard the second register, i.e., we will no longer make use of its contents. Thus we obtain the mixed state

$$\rho = \mathrm{Tr}_2\Big(U\big(|\xi\rangle\langle\xi| \otimes |0\rangle\langle 0|\big)U^\dagger\Big) = 2^{-k} \sum_{x,y:\, x-y\in D} |x\rangle\langle y|$$

After that, we apply the operator $H^{\otimes k}$ to the remaining first register. This yields a new mixed state

$$\gamma = H^{\otimes k}\rho H^{\otimes k} = 2^{-2k}\sum_{a,b}\sum_{x,y:\, x-y\in D}(-1)^{a\cdot x-b\cdot y}|a\rangle\langle b|$$

Simon's algorithm

$$\gamma = H^{\otimes k} \rho H^{\otimes k} = 2^{-2k} \sum_{a,b} \sum_{x,y:\, x-y \in D} (-1)^{a\cdot x - b\cdot y} |a\rangle\langle b|$$

$\sum_{x,y:\, x-y\in D} (-1)^{a\cdot x - b\cdot y}$ is different from zero only in the case where $a = b \in E^*$. Hence

$$\gamma = \frac{1}{|E^*|} \sum_{a\in E^*} |a\rangle\langle a|$$

This is precisely the density matrix corresponding to the uniform probability distribution on the group E^*.

Lemma

Let $h_1, \ldots, h_l$ be independent uniformly distributed random elements of an Abelian group X. Then they generate the entire group X with probability $\geq 1 - |X|/2^l$.

Lemma outcome

Therefore, $2k$ random elements suffice to generate the entire group E^* with probability of error $\leq 2^{-k}$, where “error” refers to the case where $h_1, \dots, h_{2k}$ actually generate a proper subgroup of E^*.

Note that such a small — compared to 1/3 — probability of error is obtained without much additional expense. To make it still smaller, it is most efficient to use the standard procedure: repeat all calculations several times and choose the most frequent outcome. Remember probability amplification procedure.

Summary

Summing up, to find the “hidden subgroup” D, we need $O(k)$ queries to the quantum oracle. It is proven that it is not possible with less than $O(k)$ queries. The overall complexity of the algorithm is $O(k^3)$.

Should complexity be $O(k^2)$? Answer is No:

- For one query we have $O(k)$ many operations, e.g., Hadamard.
- We repeat operations $2k$ times.
- Classical postprocessing is necessary – and it is taking more time, for example, Gaussian elimination has runtime $O(k^3)$.

Classically complexity is proportional to $O(2^{k/2})$.

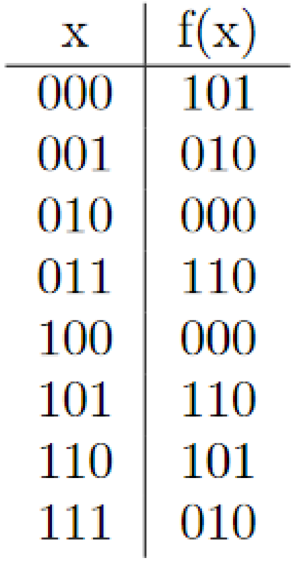
Another explanation

Elements of a group are bit strings, and hidden subgroup is our secret bit string: $s \in \{0,1\}^n$, such that:

$$f(x) = f(y) \iff x = y \oplus s$$

Here our subgroup is $\{0, s\}$.

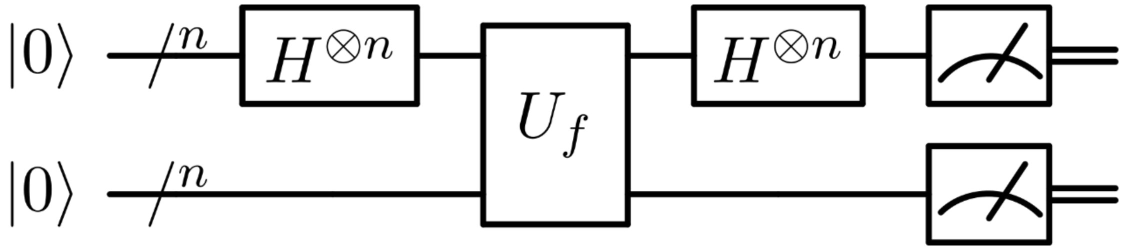
Example for $s = 110$:



x	f(x)
000	101
001	010
010	000
011	110
100	000
101	110
110	101
111	010

Another explanation

We apply the following circuit to the qubits set in state $|0\rangle$ (algorithm as described in a book, so we apply Hadamards, oracle, and Hadamards):



and get the following state before the measurement:

$$\sum_{x\in\{0,1\}^n} |x\rangle \otimes |f(x)\rangle \longmapsto \sum_{z\in\{0,1\}^n} \sum_{x\in\{0,1\}^n} (-1)^{x\cdot z}|z\rangle \otimes |f(x)\rangle$$

Another explanation

$$\sum_{x\in\{0,1\}^n} |x\rangle \otimes |f(x)\rangle \longmapsto \sum_{z\in\{0,1\}^n} \sum_{x\in\{0,1\}^n} (-1)^{x\cdot z}|z\rangle \otimes |f(x)\rangle$$

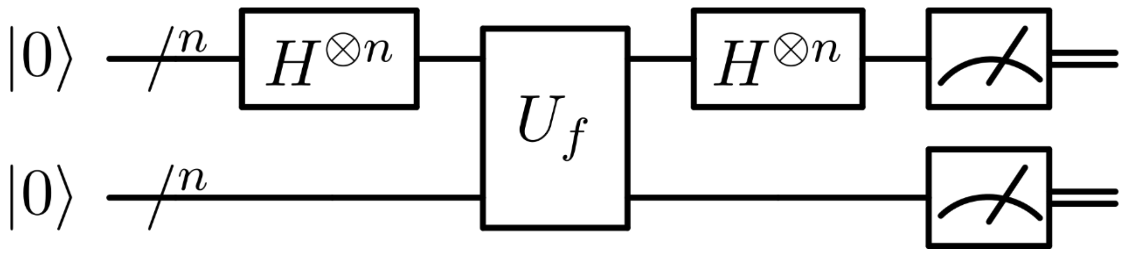
After doing the measurement, we get a bit string $x \in \{0,1\}^n$ such that it’s inner product modulo two with the secret string s is zero. This doesn’t tell us s exactly, but it tells us “something about s”.

Summary

The high-level idea behind Simon's algorithm is to "probe" (or "sample") a quantum circuit "enough times" to find $n-1$ n-bit strings, that is $y_1, \ldots, y_{n-1} \in \{0,1\}^n$ such that the following equations are satisfied

$$\begin{cases} y_1 \cdot s = 0 \\ y_2 \cdot s = 0 \\ \vdots \\ y_{n-1} \cdot s = 0 \end{cases}$$

So, this linear system contains $n-1$ linear equations in n unknowns (i.e. the bits of $s \in \{0,1\}^n$, and the goal is to solve it to obtain s, and s is fixed for a given function f. There is not always a (unique) solution.



Factoring and finding the period for raising to a power

BQP ⊃ BPP evidences

A second piece of evidence in favor of the hypothesis BQP⊃BPP is the fast quantum algorithm for factoring integers into primes and for another number-theoretic problem — finding the discrete logarithm. They were found by P. Shor. Let us discuss the first of these two problems.

Factoring

Factoring (an integer into primes). Suppose we are given a positive integer y. It is required to find its decomposition into prime factors

$$y = {p_1}^{\alpha 1} {p_2}^{\alpha 2} \ldots {p_k}^{\alpha k}$$

Even simplified cases such as $y = pq$ can be considered as the same type of tasks.

Factoring complexity

This problem is thought to be so complex that practical cryptographic algorithms are based on the hypothetical difficulty of its solution. From the theoretical viewpoint, the situation is somewhat worse: there is neither a reduction of problems of class NP to the factoring problem, nor any other “direct” evidence in favor of its complexity. (The word “direct” is put in quotation marks because at present the answer to the question P=NP is unknown.)

Factoring complexity

Therefore, the conjecture about the complexity of the factoring problem complements the abundant collection of unproved conjectures in the computational complexity theory. It is desirable to decrease the number of such problems. Shor’s result is a significant step in this direction: if we commit an “act of faith” and believe in the complexity of the factoring problem, then the need for yet another act of faith (regarding the greater computational power of the quantum computer) disappears.

Reduction to period finding

We will construct a fast quantum algorithm for solving not the factoring problem, but another problem, called Period finding, to which the factoring problem is reduced with the aid of a classical probabilistic algorithm.

In other words, we can classically reduce factoring problem to period finding, and then solve it quantumly. Reduction means – if we are able to solve efficiently period finding, then we can efficiently solve factoring.

Period finding

Suppose we are given an integer $q > 1$ that can be written using at most n binary digits (i.e., $q < 2^n$) and another integer a such that $1 \leq a < q$ and $\gcd(a, q) = 1$ (where $\gcd(a, q)$ denotes the greatest common divisor). It is required to find the period of a with respect to q, i.e., the smallest nonnegative number t such that $a^t \equiv 1 (mod\ q)$.

Period finding

In other words, the period is the order of the number a in the multiplicative group of residues $(Z/qZ)^*$. We will denote the period of a with respect to q by $per_q\ (a)$.

Example for $q = 7, a = 3$, period $t = 6$:

$$\begin{aligned}
3^1 &= 3 &= 3^0 \times 3 &\equiv 1 \times 3 &= 3 &\equiv 3 \pmod 7 \\
3^2 &= 9 &= 3^1 \times 3 &\equiv 3 \times 3 &= 9 &\equiv 2 \pmod 7 \\
3^3 &= 27 &= 3^2 \times 3 &\equiv 2 \times 3 &= 6 &\equiv 6 \pmod 7 \\
3^4 &= 81 &= 3^3 \times 3 &\equiv 6 \times 3 &= 18 &\equiv 4 \pmod 7 \\
3^5 &= 243 &= 3^4 \times 3 &\equiv 4 \times 3 &= 12 &\equiv 5 \pmod 7 \\
3^6 &= 729 &= 3^5 \times 3 &\equiv 5 \times 3 &= 15 &\equiv 1 \pmod 7 \\
3^7 &= 2187 &= 3^6 \times 3 &\equiv 1 \times 3 &= 3 &\equiv 3 \pmod 7
\end{aligned}$$

Next lectures

In next lectures we will examine a quantum algorithm for the solution of the period finding problem. But we will begin by describing the classical probabilistic reduction of the factoring problem to the problem of finding the period. We suggest to revise the probabilistic test for primality that we discussed in Lecture 7.

Primality testing algorithm

First steps

Assume that a number q is given.

Step 1. If q is even (and $q \neq 2$), then q is composite. If q is odd, we proceed to Step 2.

Step 2. Let $q - 1 = 2^k l$, where $k > 0$, and l is odd.

Step 3. We choose a random $a \in \{1, \ldots, q - 1\}$.

Step 4. We compute $a^l, a^{2l}, a^{4l}, \ldots, a^{2^k l} = a^{q-1}$ modulo q.

Two tests

Test 1. If $a^{q-1} \neq 1$ (where modular arithmetic is assumed), then q is composite.

Test 2. If the sequence $a^l, a^{2l}, \dots, a^{2^k l}$ (Step 4) contains a 1 that is preceded by anything except ± 1, then q is composite. In other words, if there exists j such that $a^{2^j l} \neq \pm 1$ but $a^{2^{j+1} l} = 1$, then q is composite.

In all other cases the algorithm says that "q is prime" (though in fact it is not guaranteed).

Before next lecture

Chinese remainder theorem

The Chinese remainder theorem states that if one knows the remainders of the Euclidean division (division with quotient and remainder) of an integer n by several integers, then one can determine uniquely the remainder of the division of n by the product of these integers, under the condition that the divisors are pairwise coprime.

Chinese remainder theorem

Abstract terminology aside, Chinese remainder theorem says that the map $Z/qZ \to (Z/uZ) \times (Z/vZ)$ is one-to-one. In other words, for any u, v the system

$x \equiv a1\ (mod\ u)$,

$x \equiv a2\ (mod\ v)$

has a unique, up to (mod q)-congruence, solution.

Note, that this also is true for subsets of mentioned groups, namely: $(Z/qZ)^{*} \subseteq (Z/qZ)$.

Multiplicative group

The multiplicative group modulo n is cyclic if and only if n is 1, 2, 4, p^k or $2p^k$, where p is an odd prime and $k > 0$. For all other values of n the group is not cyclic.

By definition, the group is cyclic if and only if it has a generator g (a generating set $\{g\}$ of size one), that is, the powers $g^0, g^1, g^2, \ldots,$ give all possible residues modulo n coprime to n. A generator of such a group is called a primitive root modulo n.

Cyclic group example

Example for $p^k = 7^1$, generator $g = 3$

$$\begin{aligned}
3^1 &= 3 &= 3^0 \times 3 &\equiv 1 \times 3 &= 3 &\equiv 3 \pmod{7} \\
3^2 &= 9 &= 3^1 \times 3 &\equiv 3 \times 3 &= 9 &\equiv 2 \pmod{7} \\
3^3 &= 27 &= 3^2 \times 3 &\equiv 2 \times 3 &= 6 &\equiv 6 \pmod{7} \\
3^4 &= 81 &= 3^3 \times 3 &\equiv 6 \times 3 &= 18 &\equiv 4 \pmod{7} \\
3^5 &= 243 &= 3^4 \times 3 &\equiv 4 \times 3 &= 12 &\equiv 5 \pmod{7} \\
3^6 &= 729 &= 3^5 \times 3 &\equiv 5 \times 3 &= 15 &\equiv 1 \pmod{7} \\
3^7 &= 2187 &= 3^6 \times 3 &\equiv 1 \times 3 &= 3 &\equiv 3 \pmod{7}
\end{aligned}$$

Shor’s algorithm summary

- Pick x randomly in the range 1 to N−1, such that $gcd(x,N)$=1.
- Use order finding algorithm to find order of x (mod N), which will be denoted by r.
- If r is even, and $x^{r/2} \neq -1\,(mod\ N)$, then compute $gcd(x^{r/2}$−1,N) and $gcd(x^{r/2}$+1,N).
- Test to see if one of these is a non-trivial factor. If so return, otherwise the algorithm fails. If that is the case, repeat.

Thank you for your attention!