Theory of Computation, Spring 2024
Solutions to Test 3

Problem 1. Describe how, in the derivation of the lower bound for parallel computation time, we use the two main physical assumptions: that the volume of the sphere is proportional to the cube of its radius and that all the speeds are bounded by the speed of the light.

Solution: Suppose that a parallel algorithm takes time $T_{par}(x)$ to run. How far away from the user can the computing processors be located? The fastest speed of any process is the speed of light c. During that time, any signal can only travel the distance $R = c \cdot T_{par}$. Thus, all processors involved in computations are located inside the sphere of radius R. The volume of this sphere is

$$V = \frac{4}{3} \cdot \pi \cdot R^3 = \frac{4}{3} \cdot \pi \cdot c^3 \cdot T_{par}^3.$$

Thus, if we denote the volume of the smallest processor by ΔV, the number N_p of processors is limited by the fact that their total volume – which is at least $N_p \cdot \Delta V$ – cannot be larger than V: $N_p \cdot \Delta V \le V$, hence

$$N_p \le \frac{V}{\Delta V} = \frac{4}{3} \cdot \pi \cdot c^3 \cdot \frac{1}{\Delta V} \cdot T_{par}^3,$$

i.e., $N_p \le C \cdot T_{par}^3$ for some constant C.

We can simulate each parallel computation on a sequential computer, if we make this computer first perform all N_p first steps of each processor, then all N_p second steps, etc. Thus, when we go to a sequential computer, computation times increase by a factor of N_p. Thus, for the sequential time, we get $T_{seq} = N_p \cdot T_{par}$. We have an inequality for T_{par}, so we get $T_{seq} \le C \cdot T_{par}^3 \cdot T_{par} = C \cdot T_{par}^4$. Thus, $T_{par}^4 \ge C^{-1} \cdot T_{seq}$, and $T_{par} \ge \text{const} \cdot T_{seq}^{1/4}$.

Problem 2. Use the general algorithm to translate the formula

$$(\neg p \vee q \vee r \vee s) \,\&\, (\neg s \vee t)$$

into 3-CNF.

Solution: According to the general algorithm, we introduce a new variable r_1 whose meaning is $\neg p \vee q$, then the satisfiability of the original formula is equivalent to the satisfiability of the formula

$$(\neg p \vee q = r_1) \,\&\, (r_1 \vee r \vee s) \,\&\, (\neg s \vee t).$$

Then, we reduce the formula $\neg p \vee q = r_1$ to CNF.

The truth table for the negation N of this formula is:

p	q	r_1	$\neg p$	$\neg p \vee q$	N
0	0	0	1	1	1
0	0	1	1	1	0
0	1	0	1	1	1
0	1	1	1	1	0
1	0	0	0	0	0
1	0	1	0	0	1
1	1	0	0	1	1
1	1	1	0	1	0

Thus, the DNF form of N is

$$(\neg p \,\&\, \neg q \,\&\, \neg r_1) \vee (\neg p \,\&\, q \,\&\, \neg r_1) \vee (p \,\&\, \neg q \,\&\, r_1) \vee (p \,\&\, q \,\&\, \neg r_1).$$

Thus, the formula $\neg p \vee q = r_1$ has the following CNF form:

$$(p \vee q \vee r_1) \,\&\, (p \vee \neg q \vee r_1) \,\&\, (\neg p \vee q \vee \neg r_1) \,\&\, (\neg p \vee \neg q \vee r_1).$$

As a result we get the following 3-CNF expression:

$$(p \vee q \vee r_1) \,\&\, (p \vee \neg q \vee r_1) \,\&\, (\neg p \vee q \vee \neg r_1) \,\&\, (\neg p \vee \neg q \vee r_1) \,\&$$

$$(r_1 \vee r \vee s) \,\&\, (\neg s \vee t).$$

Problem 3–4. Reduce the satisfiability problem for the formula

$$(\neg a \vee \neg b \vee \neg c) \,\&\, (a \vee \neg b)$$

to:

- 3-coloring,
- clique,
- subset sum problem, and
- interval computations.

In all these reductions, explain what will correspond to $a = T$, $b = F$, and $c = T$.

Solution: For *3-coloring*, we do the following:

- We start with the palette, i.e., with vertices T, F, and U which are connected to each other.
- We form vertices a and $\neg a$, and we connect them to each other and to U.
- We form vertices b and $\neg b$, and we connect them to each other and to U.
- We form vertices c and $\neg c$, and we connect them to each other and to U.
- We form a new vertex $\neg a \vee \neg b$ and connect it to U.
- We form new vertices $(\neg a \vee \neg b)_1$ and $\neg c_1$, connect them to each other and to T, connect $(\neg a \vee \neg b)_1$ to $\neg a \vee \neg b$, and connect $\neg c_1$ to $\neg c$.
- We form new vertices $\neg a_1$ and $\neg b_1$, connect them to each other and to $\neg a \vee \neg b$, connect $\neg a_1$ to $\neg a$, and connect $\neg b_1$ to $\neg b$.
- We form new vertices a_2 and $\neg b_2$, connect them to each other and to T, connect a_2 to a, and connect $\neg b_2$ to $\neg b$.

When $a = T$, $b = F$, and $c = T$, we can color the vertices in the following colors:

- The vertex T is colored T, the vertex F is colored F, and the vertex U is colored with color U.
- a is colored with color T, $\neg a$ is colored with color F.
- b is colored with color F, $\neg b$ is colored with color T.
- c is colored with color T, $\neg c$ is colored with color F.
- The vertex $\neg a \vee \neg b$ is colored with color T.
- The vertex $(\neg a \vee \neg b)_1$ is colored F, the vertex c_1 is colored U.

- About the vertices $\neg a_1$ and $\neg b_1$, we have a choice: one of them can be colored by F, another one by U.
- The vertex a_2 is colored F, the vertex $\neg b_2$ is colored U.

One can check that in this case, no two connected vertices have the same color.

For *clique*, we add the following vertices: $\neg a_1$, $\neg b_1$, $\neg c_1$, a_2, and $\neg b_2$. We then:

- connect $\neg a_1$ with $\neg b_2$;
- connect $\neg b_1$ with a_2 and with $\neg b_2$; and
- connect $\neg c_1$ with a_2 and with $\neg b_2$.

The values $a = T$, $b = F$, and $c = T$ correspond to two possible 2-cliques:

- $\neg b_1$ and a_2, and
- $\neg b_1$ and $\neg b_2$.

For *subset sum*, by applying the general algorithm, we get the following table:

	a	b	c	C_1	C_2
a	1	0	0	0	1
$\neg a$	1	0	0	1	0
b	0	1	0	0	0
$\neg b$	0	1	0	1	1
c	0	0	1	0	0
$\neg c$	0	0	1	1	0
C_1'	0	0	0	1	0
C_1''	0	0	0	1	0
C_2'	0	0	0	0	1
C_2''	0	0	0	0	1
S	1	1	1	3	3

The values $a = T$, $b = F$, and $c = T$ correspond to selecting coins $a = 10001$, $\neg b = 01011$, and $c = 00100$. To get the desired sum 11133, we also need to add coins $C_1' = 00010$, $C_2' = 00010$, and $C_2'' = 00001$.

For *interval computations*, we get the polynomial

$$(1 - A \cdot B \cdot C) \cdot (1 - (1 - A) \cdot B),$$

where A, B, and C are from the interval $[0, 1]$. This polynomial attains the value 1 when $A = 1$, $B = 0$, and $C = 1$, so its largest possible value is 1.

Problem 5. Show how to compute the "or" of 12 Boolean values in parallel if we have an unlimited number of processors and we can ignore communication time. Why do we need parallel processing in the first place? If we take communication time into account, how much time do we need to compute the product of n numbers? What is NC? Give an example of a P-complete problem.

Solution:

- At moment $t = 1$, Processor 1 computes $r_1 = x_1 \vee x_2$, Processor 2 computes $r_2 = x_3 \vee x_4$, Processor 3 computes $r_3 = x_5 \vee x_6$, and Processor 4 computes $r_4 = x_7 \vee x_8$.
- At moment $t = 2$, Processor 1 computes $r_5 = r_1 \vee r_2$, Processor 2 computes $r_6 = r_3 \vee r_4$, Processor 3 computes $r_7 = x_9 \vee x_{10}$, and Processor 4 computes $r_8 = x_{11} \vee x_{12}$. As a result, we get $r_5 = x_1 \vee x_2 \vee x_3 \vee x_4$ and $r_6 = x_5 \vee x_6 \vee x_7 \vee x_8$.
- At moment $t = 3$, Processor 1 computes $r_9 = r_5 \vee r_6$ and Processor 2 computes $r_{10} = r_7 \vee r_8$. As a result, we get the values $r_9 = x_1 \vee \ldots \vee x_8$ and $r_{10} = x_9 \vee x_{10} \vee x_{11} \vee x_{12}$.
- At moment $t = 4$, Processor 1 computes the desired result $r_9 \vee r_{10}$. One can easily check that this value is equal to $x_1 \vee \ldots \vee x_{12}$.

For this computation, we need 4 processors and 4 moments of time.

For computing the product of n numbers, sequential computations require at least $T_{\text{sequential}}(n) \geq n - 1$ steps. Since $T_{\text{parallel}}(n) \geq \text{const} \cdot (T_{\text{sequential}}(n))^{1/4}$, we thus need $T_{\text{parallel}}(n) \geq c \cdot n^{1/4}$.

NC is the class of all problems that can be computed on polynomial number of processors ($N_{\text{processors}} \leq P(n)$ for some polynomial $P(n)$ of the length n of the input) in polylog time, i.e., in time bounded by $P(\log(n))$ for some polynomial $P(n)$.

An example of P-complete problem is *linear programming*: checking whether a given set of linear inequalities $a_{i1} \cdot x_1 + \ldots + a_{in} \cdot x_n \geq b_i$, where a_{ij} and b_i are known, and x_j are unknowns, has a solution.

Problem 6. What can you say about the Kolmogorov complexity of the following string: 101101... in which 101 is repeated 17,000 times.

Solution: A possible program for producing this sequence x is as follows:

```
for(int i = 1; i <= 17000; i++)
     System.out.print('101');
```

This program has $33 + 31 = 64$ symbols, so the Kolmogorv complexity $K(x)$ of this string – which is the length of the shortest program for computing this string – is smaller than or equal to 64: $K(x) \leq 64$.

Problem 7. Suppose that we have a probabilistic algorithm that gives a correct answer half of the time. How many times do we need to repeat this algorithm to reduce probability of error to at most 10%? Give an example of a probabilistic algorithm. Explain why we need probabilistic algorithms in the first place.

Solution: The probability of error is $1-1/2 = 1/2$. We want the probability of error to be smaller than $10\% = 1/10$. So, we need to select the smallest number of iterations k for which

$$\frac{1}{2^k} \le \frac{1}{10},$$

i.e., equivalently, for which $2^k \ge 10$.

- For $k = 1$, we get $2^1 = 2 < 10$.
- For $k = 2$, we get $2^2 = 4 < 10$.
- For $k = 3$, we get $2^3 = 8 < 10$.
- For $k = 4$, we get $2^4 = 16 > 10$.

Thus, we need 4 iterations.

An example of a probabilistic algorithm is an algorithm for checking program correctness. If someone proposes a program $f(x)$ that, given a real number $x \in [0, 1]$, supposed to compute the desired expression $g(x)$, then a natural way to check whether the program is correct is run a random number generator several times and check that for all the resulting random values $r_1, \ldots, r_k$, we get $f(r_i) = g(r_i)$.

We need probabilistic algorithms since many practical problems are NP-complete, which means that we cannot have a feasible algorithm that always produced a correct answer. It is thus reasonable to try to find a feasible algorithm that produces a correct answer with some probability.

Problem 8. Use the variable-elimination algorithm for checking satisfiability of the following 2-SAT formula:

$$(\neg a \vee c) \,\&\, (\neg b \vee \neg c) \,\&\, (a \vee b) \,\&\, (\neg c \vee b).$$

Find all solutions.

Solution: Let us eliminate the variable a. Clauses containing a or $\neg a$ lead to the following inequalities:

$$a \le c, \quad \neg b \le a.$$

The condition that every lower bound must be smaller than or equal than every upper bound leads to

$$\neg b \le c \text{ and } \neg b \le c.$$

This is equivalent to the clause $b \vee c$. So, for remaining variables b and c, we have the following clauses:

$$(\neg b \vee \neg c) \,\&\, (\neg c \vee b) \,\&\, (b \vee c).$$

Let us now eliminate the variables b. Clauses containing b or $\neg b$ lead to the following inequalities:

$$b \le \neg c, \quad c \le b, \quad \neg c \le b.$$

The condition that every lower bound must be smaller than or equal than every upper bound leads to the following inequalities:

$$c \le \neg c, \quad \neg c \le \neg c.$$

The second of these inequalities is always trivially true. The first one is only true when $c = 0$. Thus, $c = 0$. For $c = 0$, inequalities containing b lead to

$$b \le 1, \quad 0 \le b, \quad 1 \le b.$$

Thus, $b = 1$.

From $a \le c$, we can now conclude that $a = 0$. Thus, the only solution is $a = 0$, $b = 1$, and $c = 0$.

Problem 9. How is the negation operation $f(x_1) = \neg x_1$ – which is true when x_1 is false and false when x_1 is true – represented in quantum computing? Provide a general formula and explain it on the example when x_1 is true and the auxiliary variable is false.

Solution: the general way to represent a function $f(x_1, \ldots, x_n)$ is as a mapping

$$|x_1, \ldots, x_n, y\rangle \to |x_1, \ldots, x_n, y \oplus f(x_1, \ldots, x_n)\rangle.$$

In our case, $n = 1$ and $f(x_1) = \neg x_1$, so we have

$$|x_1, y\rangle \to |x_1, y \oplus \neg x_1\rangle.$$

In particular, when $x_1 = 1$ and $y = 0$, we get

$$|1, 0\rangle \to |1, 0 \oplus \neg 1\rangle = |1, 0 \oplus 0\rangle = |1, 0\rangle.$$

Problem 10. Briefly describe what you have done for your project.