Theory of Computation, Spring 2020
Solutions to Test 3

1. Where in the proof that propositional satisfiability is NP-hard do we use the two main physical assumptions: that all the speeds are bounded by the speed of the light and that the volume of the sphere is proportional to the cube of its radius.

Solution: These assumptions are used in two places.

First, they are used to provide a bound on the number of direct neighbors of each cell. During time Δt, any signal can move by no more than the distance $r = c \cdot \Delta t$. So, to determine the state of the cell at the next moment of time, it is sufficient to only consider, at the previous moment of time, the states of the cells which are at distance r from this cell. We call such cells direct neighbors. All these cells are within the sphere of radius r. The volume of this sphere is $\frac{4}{3} \cdot \pi \cdot r^3$. Each cell has a volume at least Δv, so overall, we can have no more than $\dfrac{\frac{4}{3} \cdot \pi \cdot r^3}{\Delta V}$ neighboring cells. What is important is that this number does not depend on the size of the input.

The second time these assumptions are used is when we estimate the length of the resulting long formula in CNF. This formula includes variables $s_{i,b,t}$, where $i \le N_{\text{cells}}$ is the number of a cell, $b \le B$ is the number of a bit, and $t \le T$ is the number of moment of time. We are simulating feasible-time computations that check $C(x, y)$, so the time T is bounded by the polynomial of the size of the input. During this time T, only cells at a distance $\le R = c \cdot T$ could affect the computation result, so all these cells are within the sphere of radius R. Its volume is $\frac{4}{3} \cdot \pi \cdot R^3$, so we can have no more than $\dfrac{\frac{4}{3} \cdot \pi \cdot R^3}{\Delta V} = \text{const} \cdot T^3$ cells. Since T is bounded by some polynomial $P(n)$, T^3 is also bounded by a polynomials $P^3(n)$, so N_{cells} is polynomial, hence the number of variables and the length of the long formula are also bounded by a polynomials.

2. Use the general algorithm to translate the formula $(\neg a \vee b \vee \neg c \vee d)\,\&\,(a \vee \neg b)$ into 3-CNF.

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

equivalent to the satisfiability of the formula

$$(\neg a \vee b = r_1)\,\&\,(r_1 \vee \neg c \vee d)\,\&\,(a \vee \neg b).$$

Then, we reduce the formula $(\neg a \vee b = r_1)$ to CNF. The negation of this formula has the form

$$(\neg a\,\&\,\neg b\,\&\neg r_1) \vee (\neg a\,\&\,b\,\&\neg r_1) \vee (a\,\&\,\neg b\,\&r_1) \vee (a\,\&\,b\,\&\neg r_1).$$

Thus, this formula has the form

$$(a \vee b \vee r_1)\,\&\,(a \vee \neg b \vee r_1)\,\&\,(\neg a \vee b \vee \neg r_1)\,\&\,(\neg a \vee \neg b \vee r_1).$$

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

$$(a \vee b \vee r_1)\,\&\,(a \vee \neg b \vee r_1)\,\&\,(\neg a \vee b \vee \neg r_1)\,\&\,(\neg a \vee \neg b \vee r_1)\,\&$$

$$(r_1 \vee \neg c \vee d)\,\&\,(a \vee \neg b).$$

3-4. Reduce the satisfiability problem for the formula $(\neg a \vee \neg b \vee c)\,\&\,(a \vee \neg b)$ to:

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

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

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 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 c_1 to 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 $\neg b_1$, connect them to each other and to T, connect a_2 to a, and connect $\neg b_2$ to $\neg b$.

When $a = T$ and $b = c = F$, 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 F, $\neg c$ is colored with color T.
- 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.
- The vertex $\neg a_1$ is colored U, the vertex $\neg b_1$ is colored F.
- For a_2 and $\neg b_2$, we have a choice: one of these vertices much be colored by color F, another one by color U.

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

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

The values $a = T$ and $b = c = F$ 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	1	0
$\neg c$	0	0	1	0	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 $a = T$ and $b = c = F$ correspond to selecting coins $a = 10001$, $\neg b = 01011$, and $\neg c = 00100$. To get the desired sum 11133, we also need to add coins $C_1' = 00010$, $C_1'' = 00010$, and $C_2'' = 00001$.

For *interval computations*, we get the polynomial

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

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

5. Show how to compute the maximum of 11 numbers 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 maximum of n numbers? What is NC? Give an example of a P-complete problem.

Solution:

- At moment $t = 1$, Processor 1 computes $r_1 = \max(x_1, x_2)$, Processor 2 computes $r_2 = \max(x_3, x_4)$, Processor 3 computes $r_3 = \max(x_5, x_6)$, Processor 4 computes $r_4 = \max(x_7, x_8)$, Processor 5 computes $r_5 = \max(x_9, x_{10})$.
- At moment $t = 2$, Processor 1 computes $r_6 = \max(r_1, r_2)$, Processor 3 computes $r_7 = \max(r_3, r_4)$, and Processor 5 computes $r_8 = \max(r_5, x_{11})$. As a result, we get $r_6 = \max(x_1, x_2, x_3, x_4)$, $r_7 = \max(x_5, x_6, x_7, x_8)$, and $r_8 = \max(x_9, x_{10}, x_{11})$.
- At moment $t = 3$, Processor 1 computes $r_9 = \max(r_6, r_7)$. As a result, we get the value $r_9 = \max(x_1, \ldots, x_8)$.
- At moment $t = 4$, Processor 1 computes the desired result $\max(r_9, r_8)$. One can easily check that this value is equal to $\max(x_1, \ldots, x_{11})$.

For this computation, we need 5 processors and 4 moments of time.

For computing the maximum 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 $T_{\text{parallel}}(n) \geq c \cdot (n - 1)^{1/4}$.

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.

6. What can you say about the Kolmogorov complexity of the following string: 010110111...011..1... all the way to 0 followed by 4000 1s.

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

```
for(int i = 1; i <= 4000; i++)
  {System.out.print('0');
   for(int j = 1; j <= i; j++)
     {System.out.print('1');}}
```

Even if we count blank spaces at the beginning of each line and between symbols, we get at most 32 characters on each line. With 4 lines, this means that the length of this program is smaller than or equal to $4 \cdot 32 = 128$, so $K(x) \leq 128$.

7. Suppose that we have a probabilistic algorithm that gives a correct answer 75% of the time. How many times do we need to repeat this algorithm to reduce probability of error to at most 0.1%? Give an example of a probabilistic algorithm. Explain why we need probabilistic algorithms in the first place.

Solution: The probability of error is $1 - 0.75 = 1/4$. We want the probability of error to be smaller than $0.1\% = 10^{-3}$. So, we need to select the smallest number of iterations k for which $1/4^k < 10^{-3}$.

- For $k = 2$, we get $1/4^2 = 1/16 > 1/1000$.
- For $k = 3$, we get $1/4^3 = 1/64 > 1/1000$.
- For $k = 4$, we get $1/4^4 = 1/256 > 1/1000$.
- For $k = 5$, we get $1/4^5 = 1/1024 < 1/1000$.

Thus, we need 5 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.

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

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

Find all solutions.

Solution: Let us eliminate the variable a. Clauses containing a or $\neg a$ lead to the following inequalities: $a \leq \neg b$, $a \leq \neg c$, $a \leq b$, and $\neg b \leq a$. The condition that every lower bound must be smaller than or equal than every upper bound leads to $\neg b \leq \neg c$ and $\neg b \leq b$. The last inequality is only true when $b = 1$, then $\neg b \leq a \leq b, \neg b$ implies that $0 \leq a \leq 0$, i.e., that $a = 0$.

Substituting $b = 1$ into the remaining clauses $(\neg b \vee \neg c)\,\&\,(\neg c \vee b)$, we get $\neg c$. This has to be true, so $c = 0$. Thus, the only solution is $a = 0$, $b = 1$, and $c = 0$.

9. How is "or"-operation represented in quantum computing? Provide a general formula and explain it on the example when one of the inputs is true, and another one 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, $n = 2$ and $f(x_1, x_2) = x_1 \vee x_2$, so we have

$$|x_1, x_2, y\rangle \to |x_1, x_2, y \oplus (x_1 \vee x_2)\rangle.$$

In particular, when $x_1 = 0$, $x_2 = 1$, and $y = 1$, we get

$$|0, 1, 1\rangle \to |0, 1, 1 \oplus (0 \vee 1)\rangle = |0, 1, 1 \oplus 1\rangle = |0, 1, 0\rangle.$$

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