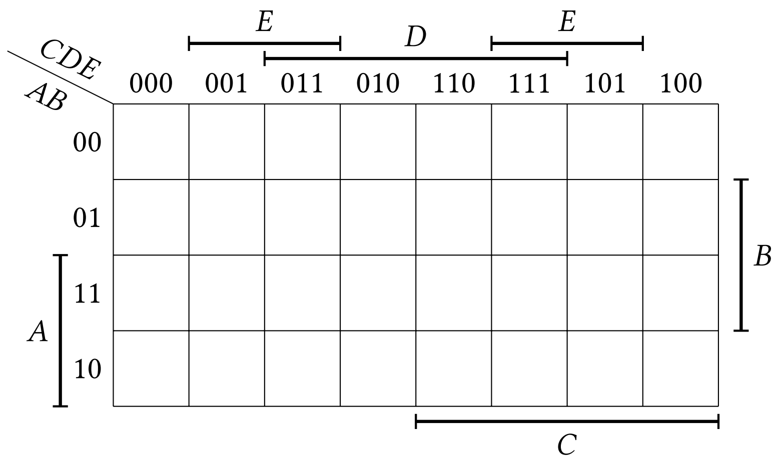
1. Perform the following steps:
   (a) Calculate the SHA-256 hash h of the string s = "DM Fall 2023 HW3" (without quotes, with all spaces, encoded in UTF-8). Convert hash h to a 256-bit binary string b (prepend leading zeros if necessary). Cut the binary string b into eight 32-bit slices $r_1, \dots, r_8$, *e.g.* $r_2 = b_{33..64}$. Xor all slices into a 32-bit string $d = r_1 \oplus \cdots \oplus r_8$. Compute $w = d \oplus \texttt{0x24d03294}$.
   Hint: last (least significant) bits of h are $\dots\texttt{01001001}$, last bits of d are $\dots\texttt{0001}$.
   (b) Draw the Karnaugh map (use a template below) for a function $f(A, B, C, D, E)$ defined by the truth table $w = (w_1 \dots w_{32})$, where MSB corresponds to $f(\mathbb{0}) = w_1$ and LSB to $f(\mathbb{1}) = w_{32}$.
   (c) Use K-map to find the minimal DNF and minimal CNF for the function f.
   (d) Use K-map to find the number of prime implicants, *i.e.* the size of BCF.



2. For each given function f_i of 4 arguments, draw the Karnaugh map and use it to find BCF, minimal DNF, and minimal CNF. Additionally, construct ANF (Zhegalkin polynomial) using either the K-map, the tabular ("triangle") method or the Pascal method — use each method at least once.

> **Note:** WolframAlpha interprets the query "n-th Boolean function of k variables" in a reverse manner. In order to employ WolframAlpha properly, manually flip the truth table beforehand, *e.g.* the correct query for $f_{10}^{(2)}$ is "5th Boolean function of 2 variables", which gives $f_{10}^{(2)} = \neg x_2$, since $\texttt{rev}(1010_2) = 0101_2 = 5_{10}$.

   (a) $f_1 = f_{47541}^{(4)}$
   (b) $f_2 = \sum m(1, 4, 5, 6, 8, 12, 13)$
   (c) $f_3 = f_{51011}^{(4)} \oplus f_{40389}^{(4)}$
   (d) $f_4 = AB\overline{B}D + \overline{A}\,\overline{C}D + \overline{B}C\overline{D} + A\overline{C}D$

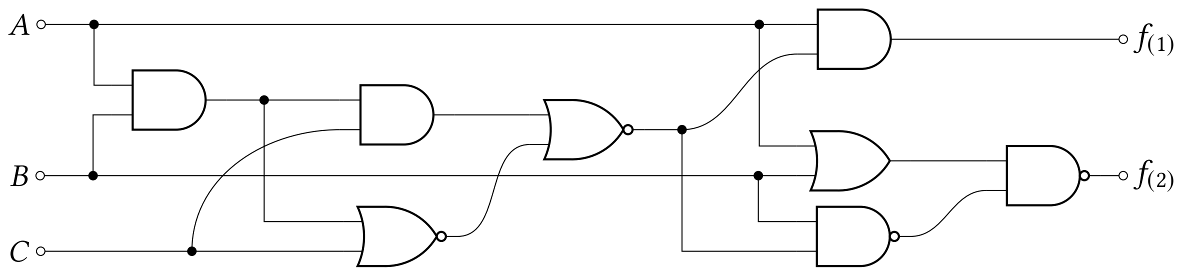
3. Convert the following formulae to CNF.
   (a) $X \leftrightarrow (A \land B)$
   (b) $Z \leftrightarrow \bigvee_i C_i$
   (c) $D_1 \oplus \cdots \oplus D_n$
   (d) $\text{majority}(X_1, X_2, X_3)$ [1]
   (e) $R \to (S \to (T \to \bigwedge_i F_i))$
   (f) $M \to (H \leftrightarrow \bigvee_i D_i)$

4. For each given system of functions F_i, determine whether it is functionally complete using Post's criterion. For each basis F_i, use it to represent the majority(A, B, C) function. Draw a combinational Boolean circuit for each resulting formula.
   (a) $F_1 = \{\land, \lor, \neg\}$
   (b) $F_2 = \{f_{14}^{(2)}\}$
   (c) $F_3 = \{\to, \not\to\}$
   (d) $F_4 = \{1, \leftrightarrow, \land\}$

5. Show — without using Post's criterion — that the Zhegalkin basis $\{\oplus, \land, 1\}$ is functionally complete.

[1] Majority function is a Boolean function that is 1 iff the majority (more than half) of the inputs are 1.

6. Compute the truth table for the function $f\colon \mathbb{B}^3 \to \mathbb{B}^2$ (with the semantics $\langle A, B, C\rangle \mapsto \langle f_{(1)}, f_{(2)}\rangle$) represented with the following circuit.



7. Construct a minimal Boolean circuit that implements the conversion of 4-bit binary numbers to Gray code, *i.e.* the function $f\colon \mathbb{B}^4 \to \mathbb{B}^4$ with the semantics $(b_3, b_2, b_1, b_0) \mapsto (g_3, g_2, g_1, g_0)$, *e.g.*, $0000_2 \mapsto 0000_{\text{Gray}}$, and $1001_2 \mapsto 1101_{\text{Gray}}$. Use only NAND and NOR logic gates.

8. A *half subtractor* is a circuit that has two bits as input and produces as output a difference bit and a borrow. A *full subtractor* is a circuit that has two bits and a borrow as input, and produces as output a difference bit and a borrow.
   (a) Construct a circuit for a half subtractor using AND gates, OR gates, and inverters.
   (b) Construct a circuit for a full subtractor using half subtractors and NAND gates.
   (c) Construct a circuit that computes the *saturating* difference of two four-bit integers $(x_3x_2x_1x_0)_2$ and $(y_3y_2y_1y_0)_2$ using half/full subtractors, AND gates, OR gates, and inverters. When $x \geq y$, the output bits $d_3, \dots, d_0$ should represent $d = x - y$, and when $x < y$, the output must be zero.

9. Construct a circuit that compares the two-bit integers $(x_1x_0)_2$ and $(y_1y_0)_2$, and outputs 1 when $x > y$ and 0 otherwise.

10. Construct a circuit that computes the product of the two-bit integers $(x_1x_0)_2$ and $(y_1y_0)_2$. The circuit should have four-bit output $(p_3p_2p_1p_0)_2$ representing the product $p = x \cdot y$.

11. Consider a Boolean function ITE: $\mathbb{B}^3 \to \mathbb{B}$ defined as follows: $\text{ITE}(c, x, y) = \left\{\begin{smallmatrix} x & \text{if } c=0 \\ y & \text{if } c=1 \end{smallmatrix}\right.$. Construct a formula for it using the standard Boolean basis $\{\wedge, \vee, \neg\}$. Determine whether the set $\{\text{ITE}\}$ is functionally complete.

12. For each given function f_i, construct a Reduced Ordered Binary Decision Diagram (ROBDD) using the natural variable order $x_1 \prec x_2 \prec \cdots \prec x_n$. Determine whether the ROBDD can be reduced even further by using a different variable order — if so, show it.

> Binary Decision Diagram (BDD) is a representation of a Boolean function as a directed acyclic graph, which consists of *decision* nodes and two *terminal* nodes (0 and 1). Each decision node is labeled by a Boolean variable x_i and has two child nodes called *low* and *high*. The edge from node to a low (high) child represents an assignment of the value `FALSE` (`TRUE`) to variable x_i. A path from the root node to the 1-terminal (0-terminal) corresponds to an assignment for which the represented Boolean function is true (false).
> BDD is called *ordered* if variables appear in the same order on all paths from the root.
> BDD is called *reduced* if it does not contain a node v with $\text{high}(v) = \text{low}(v)$, and there does not exist a pair of nodes u, v such that the sub-OBDDs rooted in u and v are isomorphic.

   (a) $f_1(x_1, \dots, x_4) = x_1 \oplus x_2 \oplus x_3 \oplus x_4$
   (b) $f_2(x_1, \dots, x_5) = \text{majority}(x_1, \dots, x_5)$
   (c) $f_3(x_1, \dots, x_4) = \sum m(1, 2, 5, 12, 15)$
   (d) $f_4(x_1, \dots, x_6) = x_1x_4 + x_2x_5 + x_3x_6$