```python
# parse.py

from pyeda.boolalg.bdd import (bddvar, expr2bdd, bdd2expr)
from pyeda.boolalg.expr import exprvar
from functools import reduce
import math

def bdd_file_parse(f):
    """
    Parse given file f, assuming it is in the correct BDD format;
    translating graph edges to Boolean formulas and BDDs
    """

    # Parse all edge node numbers
    with open(f) as bdd_file:
        edge_nodes = list(map(lambda x: list(map(int, x.split())), \
            bdd_file.readlines()))

    # Get number of unique nodes
    nodes = set()
    for en in edge_nodes:
        nodes.add(en[0])
        nodes.add(en[1])
    n = len(nodes)
    k = math.ceil(math.log(n, 2))

    # Convert each number to binary
    expr_var_names = ["v{}".format(i) for i in range(2 * k)] # for two nodes
    edge_expressions = list(map(lambda x: edge_expr(x, k, expr_var_names), \
        edge_nodes))

    # Connect all the expressions and create a bdd
    rr = None
    for r in edge_expressions:
        if rr is None: rr = expr2bdd(r)
        else: rr = rr | expr2bdd(r)

    return rr, k

def edge_expr(edge_nodes, k, expr_var_names):
    """
    Get BDD expression from each set of nodes for any given edge; e.g.,
    first edge 0-->1, i.e., 00 --> 01
    r = ~x1 & ~x2 & ~y1 & y2
    """

    # Each BDD variable gets a unique id (w/in the pairs of nodes for the edge)
    var_id = 0

    # For each node of edge, convert each bit to a BDD variable in an expression
    r = None
    for n in edge_nodes:

        node_bin = "{0:0b}".format(n)
        node_bin = "0" * (k - len(node_bin)) + node_bin
        node_bin_list = list(node_bin)

        for node_var_str in node_bin_list:

            # Give each variable a unique name, "v[unique num]"
            uniq_var_id = expr_var_names[var_id]
            var_id += 1

            # Create variable from bit
            node_var = exprvar(uniq_var_id)
            if node_var_str == "0": node_var = ~node_var

            # Update the edge relation
            if r is None: r = node_var
            else: r &= node_var

    return r
```