```
#!/usr/bin/env python3
"""
main.py
CPLEX LP file creator
Isaac Foster, Zachary Sanson
"""

import sys

FILENAME = "Path"
PATH_SPLIT = 2


def demand_eq(i, j): return 2 * i + j


def create_LP(x, y, z):
    """
    :param x: Number of total source nodes
    :param y: Number of total transit nodes
    :param z: Number of total destination nodes
    Creates a CPLEX LP file by running through all the sub-functions to create the required calculations,
constraints,
    capacities, etc.
    """
    with(open((FILENAME + str(y) + ".lp"), "w+")) as file:
        file.write("Minimize\n"
                   "\tr\n"
                   "Subject to\n"
                   "\tDemand Volume:{0}\n"
                   "\tCapacity ST:{1}\n"
                   "\tCapacity TD:{2}\n"
                   "\tTransit nodes:{3}\n"
                   "\tBinary Variables:{4}\n"
                   "\tDemand Flow:{5}\n"
                   "Bounds{6}\n"
                   "Binaries{7}\n"
                   "End".format(demand_volume(x, y, z), source_transit_capacity(x, y, z),
                                transit_destination_capacity(x, y, z), transit_nodes(x, y, z),
                                binary_variables(x, y, z), demand_flow(x, y, z),
                                bounds(x, y, z), binary(x, y, z)))


def demand_volume(x, y, z):
    """
    :param x: Number of total source nodes
    :param y: Number of total transit nodes
    :param z: Number of total destination nodes
    :return: String of demand volume constraints
    Function to return the demand volume constraint.
    That is the sum of the load between source i and Destination j:
    expected output:  x=1,y=3,z=3
                      (xikj = hij)
                      x111 + x121 + x131 = 2
                      x112 + x122 + x132 = 3
                      x113 + x123 + x133 = 4
                      ....
    """
    string = ""
    for i in range(1, x + 1):
        for j in range(1, z + 1):
            dv = []
            for k in range(1, y + 1):
                dv.append("x{0}{1}{2}".format(i, k, j))
            string += "\n\t\t" + " + ".join(dv) + " = {0}".format(demand_eq(i, j))
    return string


def transit_nodes(x, y, z):
    """
    :param x: Number of total source nodes
    :param y: Number of total transit nodes
    :param z: Number of total destination nodes
    :return: String of transit node constraints
    """
    string = ""
```

```
    for k in range(1, y + 1):
        tn = []
        for j in range(1, z + 1):
            for i in range(1, x + 1):
                tn.append("x{0}{1}{2}".format(i, k, j))
        string += "\n\t\t" + " + ".join(tn) + " - r <= 0"
    return string


def source_transit_capacity(x, y, z):
    """
    :param x: Number of total source nodes
    :param y: Number of total transit nodes
    :param z: Number of total destination nodes
    :return: String of source to transit capacities
    """
    string = ""
    for i in range(1, x + 1):
        for k in range(1, y + 1):
            st_cap = []
            for j in range(1, z + 1):
                st_cap.append("x{0}{1}{2}".format(i, k, j))
            string += "\n\t\t" + " + ".join(st_cap) + " - c{0}{1} <= 0".format(i, k)
    return string


def transit_destination_capacity(x, y, z):
    """
    :param x: Number of total source nodes
    :param y: Number of total transit nodes
    :param z: Number of total destination nodes
    :return: String of transit to destination capacities
    """
    string = ""
    for k in range(1, y + 1):
        for j in range(1, z + 1):
            td_cap = []
            # string += "\n\t\t"
            for i in range(1, x + 1):
                td_cap.append("x{0}{1}{2}".format(i, k, j))
            string += "\n\t\t" + " + ".join(td_cap) + " - d{0}{1} <= 0".format(k, j)
    return string


def binary_variables(x, y, z):
    """
    :param x: Number of total source nodes
    :param y: Number of total transit nodes
    :param z: Number of total destination nodes
    :return: String of binary variables
    Check whether the demand flow is used over a given path or not.
    Path will equal 0 if not used, or 1 if the path is loaded.
    """
    string = ""
    for i in range(1, x + 1):
        for j in range(1, z + 1):
            bv_cap = []
            for k in range(1, y + 1):
                bv_cap.append("u{0}{1}{2}".format(i, k, j))
            string += "\n\t\t" + " + ".join(bv_cap) + " = {0}".format(PATH_SPLIT)
    return string


def demand_flow(x, y, z):
    """
    :param x: Number of total source nodes
    :param y: Number of total transit nodes
    :param z: Number of total destination nodes
    :return: String of demand flow constraints
    """
    string = ""
    for i in range(1, x + 1):
        for j in range(1, z + 1):
            for k in range(1, y + 1):
                string += "\n\t\t{0} x{1}{2}{3} - {4} u{5}{6}{7} = 0".format(PATH_SPLIT, i, k, j,
                                                                             demand_eq(i, j), i, k, j)
```

```
    return string
```

```
def bounds(x, y, z):
    """
    :param x: Number of total source nodes
    :param y: Number of total transit nodes
    :param z: Number of total destination nodes
    :return: String of bounds
    Constraint to ensure all variables that have been introduced
    meet the non-negativity constraints.
    """
    string = ""
    for i in range(1, x + 1):
        for j in range(1, x + 1):
            for k in range(1, y + 1):
                string += "\n\t0 <= x{0}{1}{2}".format(i, k, j)
    return string + "\n\t0 <= r"
```

```
def binary(x, y, z):
    """
    :param x: Number of total source nodes
    :param y: Number of total transit nodes
    :param z: Number of total destination nodes
    :return: String of node combinations
    Creates a string of all possible paths.
    """
    string = ""
    for i in range(1, x + 1):
        for k in range(1, y + 1):
            for j in range(1, z + 1):
                string += "\n\tu{0}{1}{2}".format(i, k, j)
    return string


# Main function for which the program initially starts from
if __name__ == '__main__':
    # x, y, z = map(int, sys.argv[1:])    #Use if starting with terminal parameters
    # Appendix for report
    # create_LP(3, 2, 4)
    for y in range(3,10):
        create_LP(9, y, 9)
```