```
from random import randint, seed, random
from math import exp
import numpy as np
from numpy.random import poisson
from argparse import ArgumentParser


class Vertex:
    def __init__(self, l=None, r=None, ind=None, gen=None):
        self.l = l
        self.r = r
        self.ind = ind
        self.gen = gen
        self.mut = 0

    def __str__(self):
        if self.l is None:
            return str(self.ind)

        return "(%s:%d, %s:%d)" % (self.l.__str__(), self.gen -
self.l.gen,
                                   self.r.__str__(), self.gen -
self.r.gen)


class Tree:
    def __init__(self, root, total_mut, n):
        self.root = root
        self.total_mut = total_mut
        self.n = n

    @classmethod
    def simulate_tree(cls, n, N, theta, alpha):
        def pseudogeometric(N_start, ealpha, size):
            k = 0
            count = .5 * size * (size - 1)
            s = r = count / N_start
            beta = random()
            kealpha = 1
            while (beta > s):
                k += 1
                kealpha *= ealpha
                r *= (1 - count / (kealpha * N_start)) / ealpha
                s += r
            return k + 1  # as at least one generation

        N *= 2  # Diploidy
        mu = theta / (2 * N)
        ealpha = exp(-alpha)

        nodes = [Vertex(ind=i, gen=0) for i in range(n)]
        cind = n
        total_mut = 0
        while len(nodes) > 1:
            r = randint(0, len(nodes) - 1)
            l = randint(0, len(nodes) - 2)
            if l >= r:
                l, r = r, l + 1

            gen_start = max(nodes[l].gen, nodes[r].gen)
            N_start = N * ealpha**gen_start

            lp = pseudogeometric(N_start=N_start, ealpha=ealpha,
size=len(nodes))
            parent = Vertex(l=nodes[l], r=nodes[r], ind=cind,
                            gen=gen_start + lp)
            nodes[l].mut = poisson(lam=mu * (parent.gen -
nodes[l].gen), size=1)[0]
            nodes[r].mut = poisson(lam=mu * (parent.gen -
nodes[r].gen), size=1)[0]
            total_mut += nodes[l].mut + nodes[r].mut
            nodes[l] = parent
            nodes[r], nodes[-1] = nodes[-1], nodes[r]
            nodes.pop()
            cind += 1

        return cls(root=nodes[0], total_mut=total_mut, n=n)

    def __str__(self):
        return self.root.__str__()

    def tree2SNPmatrix(self):
        m = np.zeros((self.n, self.total_mut ))
        print("total_mut", self.total_mut)

        self.cur_pos = 0
        def set_mutation(node, indeces):
            new_mut = list(range(self.cur_pos, self.cur_pos +
node.mut))
            self.cur_pos += node.mut
            if node.l is None:
                m[node.ind, indeces + new_mut] = 1
            else:
                set_mutation(node.l, indeces + new_mut)
                set_mutation(node.r, indeces + new_mut)

        set_mutation(self.root, list())
        self.cur_pos = None
        return m

def main():
    parser = ArgumentParser()
    parser.add_argument("-n", dest="n", type=int, required=True)
    parser.add_argument("-N", dest="N", type=int, required=True)
    parser.add_argument("--theta", dest="theta", type=float,
required=True)
    parser.add_argument("--alpha", dest="alpha", type=float,
required=True)
```

```
    parser.add_argument("--output_tree", dest="output_tree", 
required=True)
    parser.add_argument("--output_matrix", dest="output_matrix", 
required=True)
    params = parser.parse_args()
    n, N, theta, alpha = params.n, params.N, params.theta, 
params.alpha
    print("n = %d, N = %d, theta = %f, alpha = %f" % (n, N, theta, 
alpha))
    tree = Tree.simulate_tree(n, N, theta, alpha)
    with open(params.output_tree, 'w') as f:
        print(tree, file=f)
    np.savetxt(params.output_matrix, tree.tree2SNPmatrix(), 
fmt="%d")


if __name__ == "__main__":
    main()
```