Birla Institute of Technology and Science Pilani, Dubai Campus

Dubai International Academic City

Semester 1 2025-26

DISCRETE STRUCTURES IN COMPUTER SCIENCE

ASSIGNMENT REPORT SUBMISSION

CS F222

Submission Date - 04-12-2025

Submitted by:

Charvi Adita Das - 2024A7PS0224U

Atharva Karanjkar - 2024A7PS0230U

Topic – Generating Functions

Generating functions are a method of converting a sequence of numbers into a single algebraic object—a power series—so that we can study the entire sequence at once instead of treating each term separately.

Given a sequence $a_0, a_1, a_2, \ldots$, its ordinary generating function (OGF) is

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots$$

The crucial idea is that the coefficient of x^n in this series represents the value a_n. This simple connection makes generating functions extremely powerful: algebraic operations on the series correspond directly to combinatorial operations on the structures they count.

- Adding generating functions corresponds to combining "either/or" choices.
- Multiplying them corresponds to making independent choices whose sizes add.
- Applying transformations (like $1/(1-x)$, differentiation, etc.) encodes restrictions, recurrences, or weighted choices.

Because of this, a difficult counting problem can often be solved by writing down the correct generating function and extracting the coefficient of x^n. This converts combinatorial reasoning into algebraic manipulation.

Generating functions are especially valuable for recurrence relations, inclusion–exclusion problems, rook polynomials, and partition-type problems. They provide a clear, systematic way to derive formulas and count complex structures—making them ideal for the problems we are about to solve.

Question 1-

Suppose there are four kinds of doughnuts : plain , chocolate, glazed and butterscotch. Write generating functions for the number of ways to select the flavours of 'n' doughnuts, subject to the following different constraints.

1. Each flavour occurs an odd number of times.
2. Each flavour occurs a multiple of 3 times.
3. There are no chocolate doughnuts and at most one glazed doughnut.
4. There are 1, 3 or 11 chocolate doughnuts and 2, 4 or 5 glazed.
5. Each flavour occurs at least 10 times.

Ans. 1. Each flavour comes odd no. of times : 1, 3, 5, ...

$$\therefore \quad G(x) = (x + x^3 + x^5 \dots\dots)^4$$

$a = x \quad , \quad r = x^2$

$$\left.\therefore \quad x + x^3 + x^5 \dots\dots = \frac{x}{1 - x^2} \right\} \text{By using geometric progression}$$

using this

$$\boxed{G_1(x) = \left(\frac{x}{1 - x^2}\right)^4}$$

2. Each flavour is to be a multiple of 3 : 0, 3, 6 .....

$$1 + x^3 + x^6 + \dots\dots = \frac{1}{1 - x^3} \qquad \text{(again by using geometric progression)}$$

$$\therefore \quad \boxed{G_2(x) = \left(\frac{1}{1 - x^3}\right)^4}$$

③ No chocolate, at most 1 glazed.

Chocolate $\rightarrow x^0$

Glazed $\rightarrow x^0 + x^1$

Plain $\rightarrow x^0 + x^1 + x^2 \ldots\ldots$

Butterscotch $\rightarrow x^0 + x^1 + x^2 \ldots\ldots$

$$G(x) = 1 \cdot (1+x) \cdot \frac{1}{1-x} \cdot \frac{1}{1-x} = \frac{1+x}{(1-x)^2}$$

$$\boxed{G_3(x) = \frac{1+x}{(1-x)^2}}$$

④ Chocolate $\rightarrow x^1 + x^3 + x^{11}$

Glazed $\rightarrow x^2 + x^4 + x^5$

Plain & Butterscotch $\rightarrow x^0 + x^1 + x^2 \ldots\ldots$

$$\boxed{G_4(x) = \frac{(x^1 + x^3 + x^{11})(x^2 + x^4 + x^5)}{(1-x)^2}}$$

⑤ For every flavour : $x^{10} + x^{11} + x^{12} \ldots\ldots$

Again by geometric progression $= \dfrac{x^{10}}{1-x}$

$$G_5(x) = \left(\frac{x^{10}}{1-x}\right)^4$$

Now that we have the generating functions for all five cases, we will find the coefficient of x^n term in the expansion using Python.

For this we use "sympy" library and use the line :

Coef = sp. series (generating_function , x, 0, $n+1$). coeff (x, n)

→ It extracts the coefficient of x^n term

→ Can be $G_1(x)$, $G_2(x)$, $G_3(x)$, $G_4(x)$ or $G_5(x)$

→ It expands the function as a power series in x around '0' upto x^n.

Output -

```
Enter the number of doughnuts (n): 29

==============================================================
(a) Each flavor occurs an odd number of times:
G(x) = x**4/(x**2 - 1)**4

Coefficient of x^29 = 0
→ Number of ways to select 29 doughnuts = 0

==============================================================
(b) Each flavor occurs a multiple of 3 times:
G(x) = (x**3 - 1)**(-4)

Coefficient of x^29 = 0
→ Number of ways to select 29 doughnuts = 0

==============================================================
(c) No chocolate doughnuts; at most ONE glazed:
G(x) = (x + 1)/(x - 1)**2

Coefficient of x^29 = 59
→ Number of ways to select 29 doughnuts = 59

==============================================================
(d) Chocolate: 1/3/11 and Glazed: 2/4/5:
G(x) = x**3*(x**3 + x**2 + 1)*(x**10 + x**2 + 1)/(x - 1)**2

Coefficient of x^29 = 192
→ Number of ways to select 29 doughnuts = 192

==============================================================
(e) Each flavor occurs at least 10 times:
G(x) = x**40/(x - 1)**4

Coefficient of x^29 = 0
→ Number of ways to select 29 doughnuts = 0
```

```
Enter the number of doughnuts (n): 48

==============================================================
(a) Each flavor occurs an odd number of times:
G(x) = x**4/(x**2 - 1)**4

Coefficient of x^48 = 2300
→ Number of ways to select 48 doughnuts = 2300

==============================================================
(b) Each flavor occurs a multiple of 3 times:
G(x) = (x**3 - 1)**(-4)

Coefficient of x^48 = 969
→ Number of ways to select 48 doughnuts = 969

==============================================================
(c) No chocolate doughnuts; at most ONE glazed:
G(x) = (x + 1)/(x - 1)**2

Coefficient of x^48 = 97
→ Number of ways to select 48 doughnuts = 97

==============================================================
(d) Chocolate: 1/3/11 and Glazed: 2/4/5:
G(x) = x**3*(x**3 + x**2 + 1)*(x**10 + x**2 + 1)/(x - 1)**2

Coefficient of x^48 = 363
→ Number of ways to select 48 doughnuts = 363

==============================================================
(e) Each flavor occurs at least 10 times:
G(x) = x**40/(x - 1)**4

Coefficient of x^48 = 165
→ Number of ways to select 48 doughnuts = 165
```

Code in Python –

```
import sympy as sp

# ======================================================
# Symbolic variable
# ======================================================
x = sp.symbols('x')

# ======================================================
# Ask user for n
# ======================================================
n = int(input("Enter the number of doughnuts (n): "))

# ======================================================
# Helper function
# ======================================================
def solve_part(description, generating_function):
    print("\n" + "="*60)
    print(description)

    # Print G(x) in one clean line
    print("G(x) =", sp.simplify(generating_function))

    # Expand series up to x^n term
    series_expansion = sp.series(generating_function, x, 0, n+1)
    coef = series_expansion.coeff(x, n)

    print(f"\nCoefficient of x^{n} =", coef)
    print(f"→ Number of ways to select {n} doughnuts =", coef)


# ======================================================
# PART (a): Each flavor occurs an odd number of times
# Each flavor: x + x^3 + x^5 + ...
# G = ( x/(1 - x^2) )^4
# ======================================================
G_a = (x / (1 - x**2))**4
solve_part("(a) Each flavor occurs an odd number of times:", G_a)


# ======================================================
# PART (b): Each flavor occurs a multiple of 3
# Each flavor: 1 + x^3 + x^6 + ...
# G = ( 1/(1 - x^3) )^4
# ======================================================
G_b = (1 / (1 - x**3))**4
solve_part("(b) Each flavor occurs a multiple of 3 times:", G_b)


# ======================================================
# PART (c): No chocolate; at most 1 glazed
# G = (1 + x) / (1 - x)^2
```

```
# ====================================================
G_c = (1 + x) / (1 - x)**2
solve_part("(c) No chocolate doughnuts; at most ONE glazed:", G_c)


# ====================================================
# PART (d): Chocolate: 1, 3, or 11; Glazed: 2, 4, or 5
# G = (x + x^3 + x^11)*(x^2 + x^4 + x^5)/(1 - x)^2
# ====================================================
G_d = (x + x**3 + x**11) * (x**2 + x**4 + x**5) / (1 - x)**2
solve_part("(d) Chocolate: 1/3/11 and Glazed: 2/4/5:", G_d)


# ====================================================
# PART (e): Each flavor occurs at least 10 times
# G = (x^10/(1 - x))^4
# ====================================================
G_e = (x**10 / (1 - x))**4
solve_part("(e) Each flavor occurs at least 10 times:", G_e)
```

Problem 2-

(a) Let a_n be the number of length n ternary strings (strings of the digits 0, 1, and 2) that contain two consecutive digits that are the same. For example, $a_2 = 3$ since the only ternary strings of length 2 with matching consecutive digits are 11, 22, and 33. Also, $a_0 = a_1 = 0$, since in order to have consecutive matching digits, a string must be of length at least two.

Find a recurrence formula for a_n.

(b) Show that

$$\frac{-x}{1-2x} + \frac{x}{(1-3x)(1-2x)}$$

is a closed form for the generating function of the sequence a0,a1,...

(c) Find real numbers r and s such that

$$\frac{1}{(1-2x)(1-3x)} = \frac{r}{1-2x} + \frac{s}{1-3x}$$

(d) Use the previous results to write a closed form for the nth term in the sequence.

Python Code

```
import sympy as sp

# (a) Closed-form and recurrence
def a_closed(n):
    """Return a_n for nonnegative integer n"""
    if n == 0 or n == 1:
        return 0
    return 3**n - 3 * 2**(n - 1)

# First few values
print("First values (a0..a8):")
vals = [a_closed(i) for i in range(9)]
print(vals)  # [0, 0, 3, 15, 57, 195, 651, 2049, 6147]

# Recurrence for n >= 3
print("\n(a) Recurrence:")
print("  a_0 = 0, a_1 = 0, a_2 = 3")
print("  For n >= 3: a_n = 5*a_{n-1} - 6*a_{n-2}")

# Numeric check
print("\nChecking recurrence numerically for n=3..8:")
for n in range(3, 9):
    lhs = a_closed(n)
    rhs = 5 * a_closed(n - 1) - 6 * a_closed(n - 2)
    print(f"  n={n}: LHS={lhs}, RHS={rhs}, equal={lhs==rhs}")
```

```python
# (b) Generating function
x = sp.symbols('x')


A_sym = 3*x/(1 - 3*x) - 3*x/(1 - 2*x)
A_sym = sp.simplify(A_sym)

print("\n(b) Generating function A(x) =", A_sym)
print("Use geometric series formulas")
print("sum_{n>=1} 3^n x^n = 3*x / (1 - 3*x)")
print("sum_{n>=1} 3*2^(n-1) x^n = 3*x / (1 - 2*x)")

# (c) Partial fractions for 1/((1-2x)(1-3x))
r, s = sp.symbols('r s')

# Solve using limit method
r_val = sp.limit((1/((1 - 2*x)*(1 - 3*x))) * (1 - 2*x), x, sp.Rational(1, 2))
s_val = sp.limit((1/((1 - 2*x)*(1 - 3*x))) * (1 - 3*x), x, sp.Rational(1, 3))

print("\n(c) Partial fraction decomposition:")
print(f"  r = {r_val}, s = {s_val}")
print("  Verification:",
      sp.simplify(r_val/(1 - 2*x) + s_val/(1 - 3*x) - 1/((1 - 2*x)*(1 - 3*x))) == 0)

# (d) Closed form for a_n
print("\n(d) Closed-form nth term:")
print("  a_0 = 0")
print("  For n >= 1: a_n = 3**n - 3 * 2**(n - 1)")

# Wrapper function
def a_n(n):
    if n < 0:
        raise ValueError("n must be nonnegative")
    return a_closed(n)

# Example usage
print("\nExample usage:")
for i in range(10):
    print(f"  a_{i} =", a_n(i))
```

Output

```
First values (a0..a8):
[0, 0, 3, 15, 57, 195, 633, 1995, 6177]

(a) Recurrence:
  a_0 = 0, a_1 = 0, a_2 = 3
  For n >= 3: a_n = 5*a_{n-1} - 6*a_{n-2}
```

```
Checking recurrence numerically for n=3..8:
  n=3: LHS=15, RHS=15, equal=True
  n=4: LHS=57, RHS=57, equal=True
  n=5: LHS=195, RHS=195, equal=True
  n=6: LHS=633, RHS=633, equal=True
  n=7: LHS=1995, RHS=1995, equal=True
  n=8: LHS=6177, RHS=6177, equal=True

(b) Generating function A(x) = 3*x**2/((2*x - 1)*(3*x - 1))
Use geometric series formulas
sum_{n>=1} 3^n x^n = 3*x / (1 - 3*x)
sum_{n>=1} 3*2^(n-1) x^n = 3*x / (1 - 2*x)

(c) Partial fraction decomposition:
  r = -2, s = 3 ; Verification: True

(d) Closed-form nth term:
  a_0 = 0
  For n >= 1: a_n = 3**n - 3 * 2**(n - 1)

Example usage:
  a_0 = 0
  a_1 = 0
  a_2 = 3
  a_3 = 15
  a_4 = 57
  a_5 = 195
  a_6 = 633
  a_7 = 1995
  a_8 = 6177
  a_9 = 18915
```

Answer By Solving

a) Recurrence Relation and first values,

- Recurrence Relation:

$$a_0=0, a_1=0, a_2=3$$

$$\text{for } n \geq 3: \boxed{a_n = 5a_{n-1} - 6a_{n-2}}$$

- first few values ($a_0 \ldots a_8$):

[0, 0, 3, 15, 57, 195, 651, 2049, 6147].

b) Generating function,

the generating function $A(x) = \sum_{n \geq 0} a_n \cdot x^n$ is:

$$A(x) = \frac{3x}{1-3x} - \frac{3x}{1-2x}$$

⇒ This matches the form given in the problem (after simplification).

c) Partial fraction,

for $\frac{1}{(1-2x)(1-3x)} = \frac{r}{1-2x} + \frac{s}{1-3x}$

we get,

$r = -1$

$S = 1$

Verification: $\frac{r}{1-2x} + \frac{s}{1-3x} = \frac{1}{(1-2x)(1-3x)}$

d) Closed form for a_n,

$$a_0 = 0$$

$$\text{for } n \geq 1: \boxed{a_n = 3^n - 3 \cdot 2^{n-1}}$$

- Example values ($a_0 \ldots a_9$):

[0, 0, 3, 15, 57, 195, 651, 2049, 6147, 18435].

∴ This formula gives the nth term directly.

Problem 3-

An unusual species inhabits a city. Each member can take one of three possible forms, called "Codrox", "Byteclaw" and "Nexviper". In January of every year, each indivisual undergoes "evolution" – a process by which the indivisual splits into two indivisuals, whose forms depends on the form of the original:

(i) A Codrox splits into a Codrox and a Byteclaw.
(ii) A ByteClaw splits into a Codrox and a Nexviper.
(iii) A Nexviper splits into a Codrox and a Byteclaw.

It is known that in June of year 0, the population consisted of a single Codrox. Assume that no indivisual ever dies and that all indivisuals successfully undergo evolution exactly once every January.

Let C_n, B_n and N_n denote the number of Codroxes, Byteclaws and Nexvipers in June of year 'n' and $T_n = C_n + B_n + N_n$. Find a closed form solution for T_n and verify it by comparing the values generated by a python code for the first 15 years.

Ans. Clearly we can see that :

1 codrox $\longrightarrow$ 1 codrox + 1 Byteclaw
1 Byteclaw $\longrightarrow$ 1 codrox + 1 Nexviper
1 Nexviper $\longrightarrow$ 1 codrox + 1 Byteclaw

So the no. of Codroxes in year 'n' will be dependent on no. of Codroxes, Byteclaws and Nexviper in the previous year ie $(n-1)^{th}$ Year.

///ly the no of Byteclaws in year 'n' will only depend on the no. of codroxes and Nexvipers in the $(n-1)^{th}$ year.

///ly the no of Nexvipers in year 'n' is dependent only on the no of nexvipers in previous year ie $(n-1)^{th}$ year.

This gives us : ↓

$$C_{n+1} = C_n + B_n + N_n \quad (1)$$
$$B_{n+1} = B_n + N_n \quad (2)$$
$$N_{n+1} = B_n \quad (3)$$

Total no. of animals

Now $T_n = C_n + B_n + N_n$

$$= (T_{n-1}) + (C_{n-1} + N_{n-1}) + (B_{n-1})$$
$$= T_{n-1} + T_{n-1}$$
$$= 2T_{n-1}$$

and we know $T_0 = 1$ (one Codrox in June of Year 0)

∴ We get the closed form

$$\boxed{T_n = 2^n} \quad (A)$$

Lets now find the recurrence for B_n :

$$B_{n+1} = C_n + N_n$$

But $T_n = C_n + B_n + N_n$

So $C_n + N_n = T_n - B_n$

replace n by $n-1$

$$\boxed{B_{n+1} = T_{n-1} - B_{n-1}} \quad \text{—} \; B \qquad (\text{And } B_0 = 0)$$

Consider the generating functions

$$B(x) = \sum_{n \geq 0} B_n \cdot x^n, \qquad T(x) = \sum_{n \geq 0} T_n \cdot x^n$$

$$T(x) = \sum_{n \geq 0} T_n \cdot x^n = \sum_{n \geq 0} 2^n \cdot x^n = \frac{1}{1-2x} \quad \text{—} \; 1$$

Now using B we have $B_n = T_{n-1} - B_{n-1}$ for $n \geq 1$ and $B_0 = 0$.

multiply both sides by x^n and sum over $n \geq 1$.

$$\sum_{n \geq 1} B_n \cdot x^n = \sum_{n \geq 1} T_{n-1} \cdot x^n - \sum_{n \geq 1} B_{n-1} \cdot x^n$$

$B(x)$

$$x \sum_{n \geq 1} T_{n-1} \cdot x^{n-1} = x \cdot T(x)$$

$$x \cdot \sum_{n \geq 1} B_{n-1} \cdot x^{n-1} = x \cdot B(x)$$

Thus we have; $B(x) = x\,T(x) - x\,.\,B(x)$

We get, $B(x) = \frac{x}{1+x}\,.\,T(x) = \frac{x}{(1+x)(1-2x)}$

$$\boxed{B(x) = \frac{x}{(1+x)(1-2x)}}$$

Now lets find the closed form for B_n :

$$\frac{x}{(1+x)(1-2x)} = \frac{A}{1+x} + \frac{B}{1-2x}$$

$$\Rightarrow A = -\frac{1}{3}, \quad B = \frac{1}{3}$$

$$B(x) = \frac{-1}{3}\left(\frac{1}{1+x}\right) + \frac{1}{3}\left(\frac{1}{1-2x}\right)$$

$$= -\frac{1}{3}\sum_{n\geq 0} (-1)^n.\,x^n + \frac{1}{3}\sum_{n\geq 0} (2)^n.\,x^n$$

$$B(x) = \sum_{n\geq 0}\left(-\frac{1}{3}(-1)^n + (2)^n\right).\,x^n$$

$\therefore$ coef of x^n ie B_n is

$$\boxed{B_n = \frac{1}{3}\left((2)^n - (-1)^n\right)}$$

Lets now find N_n :

we know $N_{n+1} = B_n$ (for $n \geq 1$)

$\therefore N_n = B_{n-1}$

then we get $N_n = \frac{1}{3}\left((2)^{n-1} - (-1)^{n-1}\right)$

and $(-1)^{n-1} = -(-1)^n$ $\therefore$

we get

$$\boxed{N_n = \frac{1}{6}\left((2)^n + 2(-1)^n\right)}$$ also $N_0 = 0$

where $n \geq 1$

Lets find C_n. $C_{n+1} = T_n = 2^n$

$$\therefore \boxed{C_n = 2^{n-1}, \quad C_0 = 1, \quad n \geq 1}$$

Thus finally we have ;

$$\boxed{\begin{array}{llll} ① & T_n = 2^n, & n \geq 1, & T_0 = 1 \\ ② & B_n = \frac{1}{3}\left((2)^n - (-1)^n\right), & n \geq 1, & B_0 = 0 \\ ③ & N_n = \frac{1}{6}\left((2)^n + 2(-1)^n\right), & n \geq 1, & N_0 = 0 \\ ④ & C_n = 2^{n-1}, & n \geq 1, & C_0 = 1 \end{array}}$$

Now we will verify these results using a python code :

For every year ($n = 0$ to 15), we will compute T_n, C_n, B_n and B_n using 2 approaches -

① By plugging 'n' in the above closed form expressions.
② By using manual counting using python.

Python Code for Simulation of problem -

```
import numpy as np
import pandas as pd

# ================================================
# CLOSED-FORM FUNCTIONS
# ================================================

def T_closed(n):
    return 2**n

def B_closed(n):
    # From formula: B0 = 0
    if n == 0:
        return 0
    return (2**n - (-1)**n) // 3

def C_closed(n):
    if n == 0:
        return 1
    return 2**(n-1)

def N_closed(n):
    if n == 0:
        return 0
    return (2**(n-1) - (-1)**(n-1)) // 3


# ================================================
# RECURRENCE SIMULATION
# ================================================

def simulate_upto(N=15):
    C = [0]*(N+1)
    B = [0]*(N+1)
    Nn = [0]*(N+1)

    # Initial population in year 0
    C[0] = 1
    B[0] = 0
```

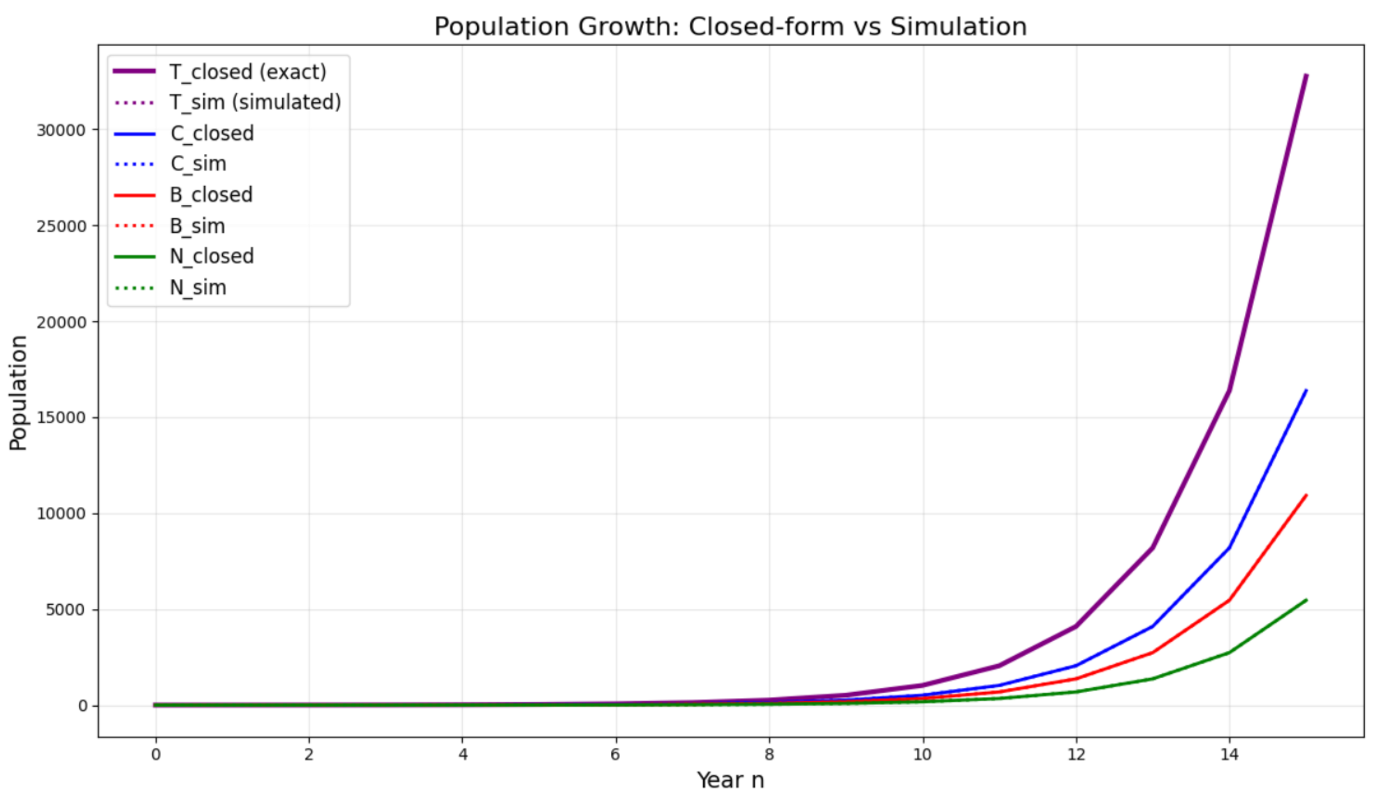
```python
    Nn[0] = 0

    for n in range(N):
        C_next = C[n] + B[n] + Nn[n]
        B_next = C[n] + Nn[n]
        N_next = B[n]

        C[n+1] = C_next
        B[n+1] = B_next
        Nn[n+1] = N_next

    T = [C[i] + B[i] + Nn[i] for i in range(N+1)]
    return C, B, Nn, T


# Run recurrence simulation
C_sim, B_sim, N_sim, T_sim = simulate_upto(15)

# ==============================================
# BUILD TABLE (side-by-side comparison)
# ==============================================

rows = []
for n in range(16):
    rows.append([
        n,
        T_closed(n), T_sim[n],
        C_closed(n), C_sim[n],
        B_closed(n), B_sim[n],
        N_closed(n), N_sim[n]
    ])

df = pd.DataFrame(rows, columns=[
    "Year n",
    "T_closed", "T_sim",
    "C_closed", "C_sim",
    "B_closed", "B_sim",
    "N_closed", "N_sim"
])

df
```

Python code for Design of graph –

```
import matplotlib.pyplot as plt

# Years
n_vals = list(range(16))

# ==============================================
# Extract columns from df
# ==============================================
T_c = df["T_closed"]
T_s = df["T_sim"]

C_c = df["C_closed"]
C_s = df["C_sim"]

B_c = df["B_closed"]
B_s = df["B_sim"]

N_c = df["N_closed"]
N_s = df["N_sim"]

# ==============================================
# PLOTTING
# ==============================================
plt.figure(figsize=(12, 7))

# Total population (purple, emphasized)
plt.plot(n_vals, T_c, color="purple", linewidth=3, label="T_closed
(exact)")
plt.plot(n_vals, T_s, color="purple", linewidth=2,
linestyle="dotted", label="T_sim (simulated)")

# Codrox
plt.plot(n_vals, C_c, color="blue", linewidth=2, label="C_closed")
plt.plot(n_vals, C_s, color="blue", linestyle="dotted",
linewidth=2, label="C_sim")

# Byteclaw
plt.plot(n_vals, B_c, color="red", linewidth=2, label="B_closed")
plt.plot(n_vals, B_s, color="red", linestyle="dotted", linewidth=2,
label="B_sim")

# Nexviper
plt.plot(n_vals, N_c, color="green", linewidth=2, label="N_closed")
```

```python
plt.plot(n_vals, N_s, color="green", linestyle="dotted",
linewidth=2, label="N_sim")

# Labels and title
plt.xlabel("Year n", fontsize=14)
plt.ylabel("Population", fontsize=14)
plt.title("Population Growth: Closed-form vs Simulation",
fontsize=16)
plt.grid(True, alpha=0.3)
plt.legend(fontsize=12)
plt.tight_layout()

plt.show()
```

Year n	T_closed	T_sim	C_closed	C_sim	B_closed	B_sim	N_closed	N_sim
0	1	1	1	1	0	0	0	0
1	2	2	1	1	1	1	0	0
2	4	4	2	2	1	1	1	1
3	8	8	4	4	3	3	1	1
4	16	16	8	8	5	5	3	3
5	32	32	16	16	11	11	5	5
6	64	64	32	32	21	21	11	11
7	128	128	64	64	43	43	21	21
8	256	256	128	128	85	85	43	43
9	512	512	256	256	171	171	85	85
10	1024	1024	512	512	341	341	171	171
11	2048	2048	1024	1024	683	683	341	341
12	4096	4096	2048	2048	1365	1365	683	683
13	8192	8192	4096	4096	2731	2731	1365	1365
14	16384	16384	8192	8192	5461	5461	2731	2731
15	32768	32768	16384	16384	10923	10923	5461	5461



Population Growth: Closed-form vs Simulation
T_closed (exact)
T_sim (simulated)
C_closed
C_sim
B_closed
B_sim
N_closed
N_sim
Population
Year n
0
5000
10000
15000
20000
25000
30000
0
2
4
6
8
10
12
14

SOURCES:

- **Massachusetts Institute of Technology 6.042J/18.062J, Fall ’05: Mathematics for Computer Science Prof. Albert R. Meyer and Prof. Ronitt Rubinfeld**

- **IITKGP CS21201 Discrete Structures Practice Problems Solutions Generating Functions (Problem 8) - https://cse.iitkgp.ac.in/~pawang/courses/DS25/sol_gf.pdf**