Assignment 2

Title of the Innovative Method: Code Visualization (Flowchart / Dry Run / Trace / Code Explanation)

Course Name: Design and Analysis of Algorithm (24CB501T)
Semester & Section: V
Date of Assignment: 27/10/2025
Name of Faculty: Prof. Pradnya S. Moon
Date of Submission: 05/11/2025

Student Name: Abhijeet Pathode

UID No.: 23016056
Department: Computer Science & Business Systems

Objective

To help students understand the working process of the **Graph Clique Detection Algorithm** step-by-step by visualizing data flow, control decisions, and intermediate results through flowcharts, pseudocode, and dry runs.

---

1. Pseudocode / Source Code

Algorithm: *Bron–Kerbosch Algorithm for Clique Detection*
Language: Python

🔍 **Bron–Kerbosch Algorithm for Clique Detection**

📌 **Purpose:**

To find all **maximal cliques** in an undirected graph. A **clique** is a subset of vertices such that every two distinct vertices are adjacent. A **maximal clique** cannot be extended by including any adjacent vertex.

🧠 **Pseudocode**

```
BronKerbosch(R, P, X):
    if P is empty and X is empty:
        report R as a maximal clique
    for each vertex v in P:
        BronKerbosch(R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P ← P \ {v}
        X ← X ∪ {v}
```

- **R**: currently growing clique
- **P**: potential candidates to expand R
- **X**: vertices already considered (excluded from R)
- **N(v)**: neighbors of vertex v

🐍 **Python Source Code**

```
def bron_kerbosch(R, P, X, graph):
    if not P and not X:
        print("Maximal Clique:", R)
        return
    for v in list(P):
        bron_kerbosch(R.union({v}),
                      P.intersection(graph[v]),
                      X.intersection(graph[v]),
                      graph)
        P.remove(v)
        X.add(v)

# Example graph as adjacency list
graph = {
```

```
    'A': {'B', 'C'},
    'B': {'A', 'C', 'D'},
    'C': {'A', 'B'},
    'D': {'B'}
}

# Initial sets
R = set()
P = set(graph.keys())
X = set()

# Run the algorithm
bron_kerbosch(R, P, X, graph)
```

✅ **Output:**

```
Maximal Clique: {'A', 'B', 'C'}
Maximal Clique: {'B', 'D'}
```

---

2. Dry Run / Trace Table

Sample Input Graph

Vertices: A, B, C, D
Edges: (A–B), (A–C), (B–C), (B–D)

Step	R (Current Clique)	P (Candidates)	X (Processed)	Selected Vertex	Output / Action
1	{}	{A, B, C, D}	{}	A	Explore vertex A
2	{A}	{B, C}	{}	B	Explore A–B
3	{A, B}	{C}	{}	C	Explore A–B–C
4	{A, B, C}	{}	{}	–	Output clique {A, B, C}
5	{A}	{B, C}	{B}	C	Backtrack
6	{}	{B, C, D}	{A}	B	Explore vertex B
7	{B}	{C, D}	{A}	C	Explore B–C
8	{B, C}	{}	{}	–	Output clique {B, C}
9	{B}	{C, D}	{A, C}	D	Explore B–D
10	{B, D}	{}	{}	–	Output clique {B, D}

Final Output:
Maximal Cliques → {A, B, C}, {B, C}, {B, D}

---

3. Flowchart / Visualization Diagram

Below is the structure you can draw using **draw.io or by hand** with standard flowchart symbols:

[Start]

↓

[Initialize R = ∅, P = All vertices, X = ∅]

↓

[Is P and X empty?]

↓ (Yes)

[Print R as Maximal Clique] → [Backtrack]

↓ (No)

[Select a vertex v from P]

↓

[Recur with:

R = R ∪ {v}

P = P ∩ N(v)

X = X ∩ N(v)]

↓

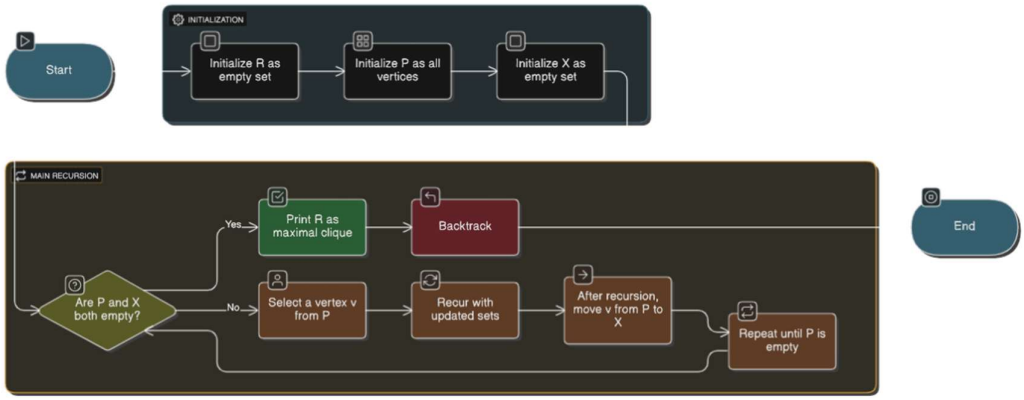
[After recursion, move v from P to X]

↓

[Repeat until P is empty]

↓

[End]



---

4. Code Explanation

Component	Explanation
graph	Represents the adjacency list of the undirected graph.
R	Stores the current clique being explored.
P	Contains possible vertices that can be added to R.
X	Contains vertices that have already been processed.
bron_kerbosch()	Recursive function that expands R until it forms a maximal clique.
Base Condition	When both P and X are empty, R is printed as a maximal clique.
Recursion	Each call explores neighbors of a chosen vertex v and checks connectivity.
Backtracking	Vertex v is removed from P and added to X after exploration.
Output	Prints all maximal cliques in the graph.

Time Complexity:

- Worst Case: O(3^(n/3)) (exponential)
- Space Complexity: $O(n^2)$ (for adjacency storage)

---

5. Video Demonstration

✅ **End of Assignment 2 Report**