Department: Computer Science and Engineering

Semester: Spring 2022

Course Title: Algorithms

Course Number: CSE246

Report on

Project: **Clique Problem**

From:

Name	**Id**
Fatema Akter	2020-1-60-115
Sheikh Fajlay Rabbi	2020-1-60-024
Abdullah al Tamim	2020-1-60-127

Instructor:

Redwan Ahmed Rizvee

Lecturer, Department of Computer Science and Engineering

East West University

Problem Statement:

The clique problem is the computational problem of finding cliques (subsets of vertices, all adjacent to each other, also called complete subgraphs) in a graph. It has several different formulations depending on which cliques, and what information about the cliques, should be found. Common formulations of the clique problem include finding a maximum clique (a clique with the largest possible number of vertices), finding a maximum weight clique in a weighted graph, listing all maximal cliques (cliques that cannot be enlarged), and solving the decision problem of testing whether a graph contains a clique larger than a given size.

Algorithm Discussion:

Our algorithm is mainly based on the clique function. Here we receive three parameters, the first parameter represents the starting node, the second parameter represents the number of vertices currently in the store array, and the third parameter is the required size of the clique. Then we check each node from the starting node that if the degree of that is greater or equal to k-1. If so then we store that vertex in the array assuming that it might become a part of the clique. After adding the vertex to the store, we check if adding that vertex forms a clique or not. If it forms a clique and the number of vertices in the store is equal to the required size of the clique then we print the clique. If it doesn't form a clique and the number of elements in the store array is less than k, then we call recursively and increase the number of elements in the store.

```
Clique(int node, int num, int k)
{
    FOR i = node+1 to v-(k-num)
        IF degree of i >= k-1
            Store[num] = i
            IF check_clique(k) = true
                IF num = k
                    Print_clique(num)
                End IF
            End IF
            IF num < k
                Clique( i, num+1, k)
            End IF
        End IF
    End FOR
}
```

Time and memory Complexity:

```
bool Check_Clique(int k)
{
    FOR i = 1 to k  ← O(k)
        FOR each vertex j ∈ Adj[i] ← O(E)
            FOR p = 1 to k ← O(k)
                IF graph[i][j] = store[p]
                    Count++
                END IF
            END FOR
        END FOR
    End FOR
    IF count == k*k-1
        return true
    Else
        return false
}
```

```
Clique(int node, int num, int k)
{
    FOR i = node+1 to v-(k-num) ← O(V)
        IF degree of i >= k-1
            Store[num] = i
            IF check_clique(k) = true
                IF num = k
                    Print_clique(num) ← O(k)
                End IF
            End IF
            IF num < k
                Clique( i, num+1, k) ← O(k)
            End IF
        End IF
    End FOR
}
```

In the check_clique function, the total running time is $O(k^2E)$. And in the clique function, the total running time is $O(V^k)$, because we make every combination of size k. Since the check_clique function is implemented inside of the clique function, so the total running is $O(k^2E * V^k)$.

And the memory complexity is O(V+E*k).

Implementation:

Clique() function: Here,

v → number of vertices.

node → starting node.

num → the number of vertices currently in the store[] array.

k → required size of the cliques.

Inside of the clique function first of all we used a for loop which starts from the starting

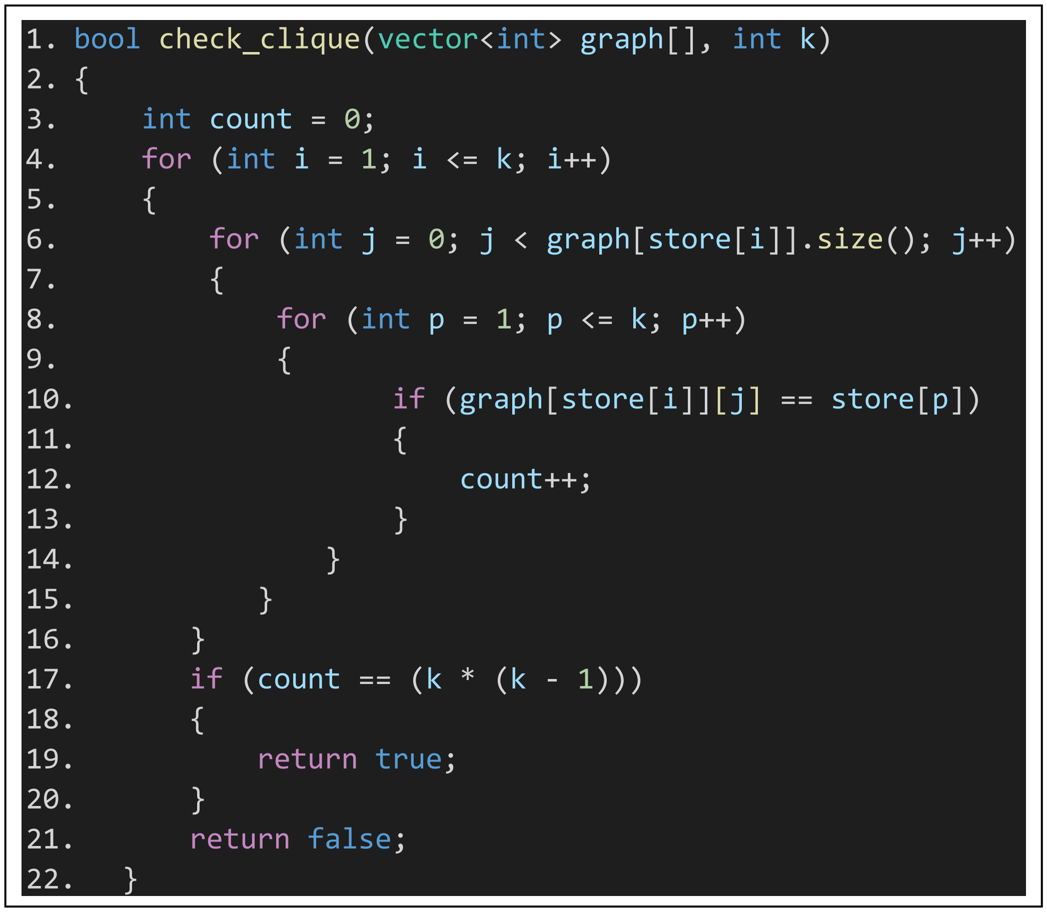
```cpp
void clique(int node, int num, int k, vector<int>
graph[])
{
    for (int i = node + 1; i <= v - (k - num); i++)
    {
        if (graph[i].size() >= k - 1)
        {
            store[num] = i;
            if (check_clique(graph, k))
            {
                if (num == k)
                    print_clique(num);
            }
            if (num < k)
                clique(i, num + 1, k, graph);
         }
   }
```

node and iterates till v-(k-num). We could iterate it just till v times but when num<k there is no need to check the last k-num number of nodes since they can't make a clique. Then we checked the degree of each vertex. If the degree is greater or equal to k-1 then we add it to the store and check if adding this vertex forms a clique or not. If the num < k then we increased the num and recursively call the function where i will be the starting node.

Check_clique() function:

Here,

k → required size of the cliques.



```cpp
bool check_clique(vector<int> graph[], int k)
{
    int count = 0;
    for (int i = 1; i <= k; i++)
    {
        for (int j = 0; j < graph[store[i]].size(); j++)
        {
            for (int p = 1; p <= k; p++)
            {
                 if (graph[store[i]][j] == store[p])
                 {
                     count++;
                 }
             }
         }
      }
      if (count == (k * (k - 1)))
      {
          return true;
      }
      return false;
   }
```

In the check_clique function, we mainly tried to iterate through all the stored elements and check if their adjacent vertices contain other stored elements or not. The size of the store[] array will not cross the size of the clique, so to traverse the store array, the loop will iterate k times. And each time if we find the stored elements in the adjacent of the ith vertex then we increment the count. Finally, if a subgraph is a clique of size k, then it will contain k*k-1 edges and the count will also be equal to k*k-1. Hence, we will return true. Otherwise, we will return false.

Print_clique():

In the print_clique function we just simply print all the elements in the store. Since all the values get stored in the array in sorted order so sorting the values again is unnecessary in our case.

```
void print_clique(int lim)
{
    for (int i = 1; i <= lim; i++)
    {
        cout << store[i] << " ";
    }
    cout << endl;
}
```

Applications:

The most important implementation of a clique is a social network where the vertices of the clique graph are present people and the edges of that graph of that clique graph represent mutual acquaintances. The clique graph represents a group of mutual friends where all of them know each other.

It can also be used in automatic test pattern generation, finding cliques can help to bound the size of a test set.