c++_report

Define the problem

The aim of this program is to find the maximal clique in a graph, where the the vertices of the graph are binary vectors of length N, and the vertices are connected if the hamming distance between them is >= d.
Our approach toward solving this problem is to first generate the combination of all possible binary vectors of length N using `generator()` function. Then, using `graph_maker` function, we create the graph by computing the hamming distance between the vertices and connecting them if the distance between them is at least d- The code iterates through all the vertices and compute their distance to eachother. We use `hamming_distance()` function to compute the hamming distance, in which its called with each iteration to compute the distance for the `graph_maker` function.
Finally, we pass the graph to `maximum_clique` function to get the maximal clique. The function iterates over all the vertices in the graph and attempts to add each vertex to a clique. At each iteration, the function checks if a given vertex can be added to the clique by checking if it is connected to all the vertices already in the clique. If the vertex can be added, it is added to the clique. This process continues until all the vertices in the graph have been checked, and the final clique with the maximum number of vertices is returned.

We will review the code function by function, explaining how the code works.
First, `Main()`:
As you see in the comments, at first we start by taking user input for the N and d values. Then we call the generator function, then we pass the generated vertices to graph_maker, after that we pass the graph to the calculating function to get the maximum clique.

```
int main() {
    // getting the input
    int N, d;
    cin >> N >> d;
    cout << "Starting..." << endl;
    // we generate the vertices
    vector<string> vertices = generator(N);

    // build the Hamming graph
    vector<set<int>> graph = graph_maker(vertices, d);

    // compute the size of the maximum clique in the graph
    int size = maximum_clique(graph);
    cout << "Size of maximum clique: " << size << endl;
```

```
    return 0;
}
```

Second, `generator()`:
In this function we generates all possible combinations of n binary digits. The first loop iterates over all possible combinations, that what makes its time complexity 2^n, the second loop iterates over the length of each combination n. Thus, the time complexity for this function is n * 2^n.

```
vector<string> generator(int n) {
    // we take the input n
    // we generate all possible combinations
    vector<string> strings;
    for (int i = 0; i < (1 << n); i++) {
        string s;
        for (int j = n - 1; j >= 0; j--) {
            s += (i >> j) & 1 ? '1' : '0';
        }
        // .push_back adds the 0/1 to the end of the string/vector etc.
        strings.push_back(s);
    }
    return strings;
}
```

Third, `graph_maker()`:
In this function, we pass the vertices and d(the minimum distance). It involves two nested For loops that iterate over all pairs of vertices in the graph. The first for loop iterates from 'i=0' to 'n' and the second for loop iterates from 'i+1' to 'n', where i is the current index of the outer loop. Thus this results in a total of n * (n-1) / 2 iterations, which is equivalent to O(n^2). We insert the index of the vertices that have a distance >=d with eachother. Example of a the code generated for n=2 and the graph for the input:

```
00
01
10
11
```

The matrix below is the graph that we generated, on the left are the indexes of the vertices, on the right side of it are the vertices that are connected to it. After making the graph, we stoped using the generated binary vectors/strings, and we started working with only indexes.

```
0|1 2 3
1|0 2 3
2|0 1 3
3|0 1 2
```

Without the index, our graph is like this:

```
1 2 3
0 2 3
0 1 3
0 1 2
```

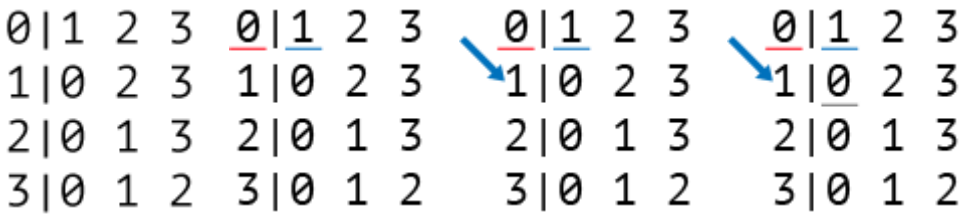
below is this function's code:

```
vector<set<int>> graph_maker(const vector<string>& vertices, int d) {
    int n = vertices.size();
    vector<set<int>> graph(n);
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (hamming_distance(vertices[i], vertices[j]) >= d) {
                // here we insert the index of the vertices that have a
distance >=d toward eachother
                // [0] 1, 2, 3
                // [1], 0, 3
                // etc.
                graph[i].insert(j);
                graph[j].insert(i);
            }
        }
    }
    return graph;
}
```

Fourth, `hamming_distance()` funtion:
This function is as important as the `main()` function in this code. In this function we calculate the distance between two vectors/vertices. The function iterates through the vector bits that are passed and calculate the distance/difference.

```
int hamming_distance(const string& a, const string& b) {
    int distance = 0;
    for (int i = 0; i < a.size(); i++) {
        if (a[i] != b[i]) {
            distance++;
        }
    }
    return distance;
}
```

Finally, `maximum_clique()` function:
In this function we iterate through the graph to find our maximal clique (i.e., the largest set of mutually connected vertices). Our approach in this function is greedy approach, we iterate over all the vertices in the graph and attempt to add the vertices to a clique. After adding the vertices to a clique, in each iteration, we compare the clique length to the previous one, and update our `max_size` integer, setting it up to the maximum of them.
To create a clique, we have two nested For loops, the first one `i` iterates through all the rows in the graph, whith each iteration we also create/add a row into a set `set<int> clique{i};`, In the second loop `for (int j: graph[i])` we iterate through the columns/items in `graph[i]`, then for each item, we check if `graph[i][j]` is in `graph[j]`-Check the attached pictures for a better understanding. If the the item is in, we add it to clique, if its not, we break the loop.



The time complexity of the `maximum_clique()` function is O(n^2), as its iterating over all the vertices in the graph and for each vertex, iterating over all the vertices connected to it.

```
int maximum_clique(const vector<set<int>>& graph){
    int max_size = 0;
    //This loop iterates over all the vertices in the graph. The graph size
is 2^n. so the iterations will be
    // 2^n, checking the connections.
    cout << "vertices: " << graph.size() << endl;
    for (int i = 0; i < graph.size(); i++) {
        // here we initialize a another set to store the connected vertices
        set<int> clique{i};
        // we iterate
        for (int j: graph[i]) {
            bool can_add = true;
            for (int k: clique) {
                if (!graph[j].count(k)) {
                    can_add = false;
                    break;
                }
            }
            if (can_add) {
                clique.insert(j);
            }
        }
        // std::max is defined in the header file <algorithm> and is used to
```

```
find out
        // the largest of the number passed to it. It returns the first of
them

        max_size = max(max_size, (int) clique.size());
    }
    return max_size;
}
```

The source code:

```
#include <algorithm>
#include <iostream>
#include <set>
#include <vector>
#include <chrono>

using namespace std;

// function to generate all possible combinations
vector<string> generator(int n) {
    // we take the input n
    // we generate all possible combinations
    vector<string> strings;
    for (int i = 0; i < (1 << n); i++) {
        string s;
        for (int j = n - 1; j >= 0; j--) {
            s += (i >> j) & 1 ? '1' : '0';
        }
        // .push_back adds the 0/1 to the end of the string/vector etc.
        strings.push_back(s);
    }
    // print the binary strings, for understanding and troubleshooting
purpose.
    /* for (const string& s : strings) {
        cout << s << endl;
    }*/
    return strings;
}

// here we compute the hamming distance between the two vectors
int hamming_distance(const string& a, const string& b) {
```

```
    int distance = 0;
    for (int i = 0; i < a.size(); i++) {
        if (a[i] != b[i]) {
            distance++;
        }
    }
    return distance;
}

// this function builds the hamming graph, it uses sets to store the
vertices, it iterates through the vertices and
// calls hamming_distance function to compute the distance
vector<set<int>> graph_maker(const vector<string>& vertices, int d) {
    int n = vertices.size();
    vector<set<int>> graph(n);
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (hamming_distance(vertices[i], vertices[j]) >= d) {
                // here we insert the index of the vertices that have a
distance >=d toward eachother
                // [0] 1, 2, 3
                // [1], 0, 3
                // etc.
                graph[i].insert(j);
                graph[j].insert(i);
            }
        }
    }
    return graph;
}

// Here we try to find the maximal clique by interating through the graph
//
int maximum_clique(const vector<set<int>>& graph){
    int max_size = 0;
    //This loop iterates over all the vertices in the graph. The graph size
is 2^n. so the iterations will be
    // 2^n, checking the connections.
    cout << "point: " << graph.size() << endl;
    for (int i = 0; i < graph.size(); i++) {
        // here we initialize a another set to store the connected vertices
        set<int> clique{i};
        // we iterate
```

```
        for (int j: graph[i]) {
            bool can_add = true;
            for (int k: clique) {
                if (!graph[j].count(k)) {
                    can_add = false;
                    break;
                }
            }
            if (can_add) {
                clique.insert(j);
            }
        }
        max_size = max(max_size, (int) clique.size());
    }
    return max_size;
}


// Our main function
// what we do is, first generate the combinations, then make the hamming
graph, then calculate the maximal.
// we also have a timer to check our run time.
int main() {
    // getting the input
    int N, d;
    cin >> N >> d;
    cout << "Starting..." << endl;
    // we generate the vertices
    vector<string> vertices = generator(N);

    // build the Hamming graph
    vector<set<int>> graph = graph_maker(vertices, d);
    // we print the graph, for understanding and troubleshooting purpose.
    /*for (int i = 0; i < graph.size(); i++) {
            for (auto x : graph[i])
                cout << x << " ";
            cout << endl;
        }*/

    // compute the size of the maximum clique in the graph
    int size = maximum_clique(graph);
    cout << "Size of maximum clique: " << size << endl;
```

```
    return 0;
}
```