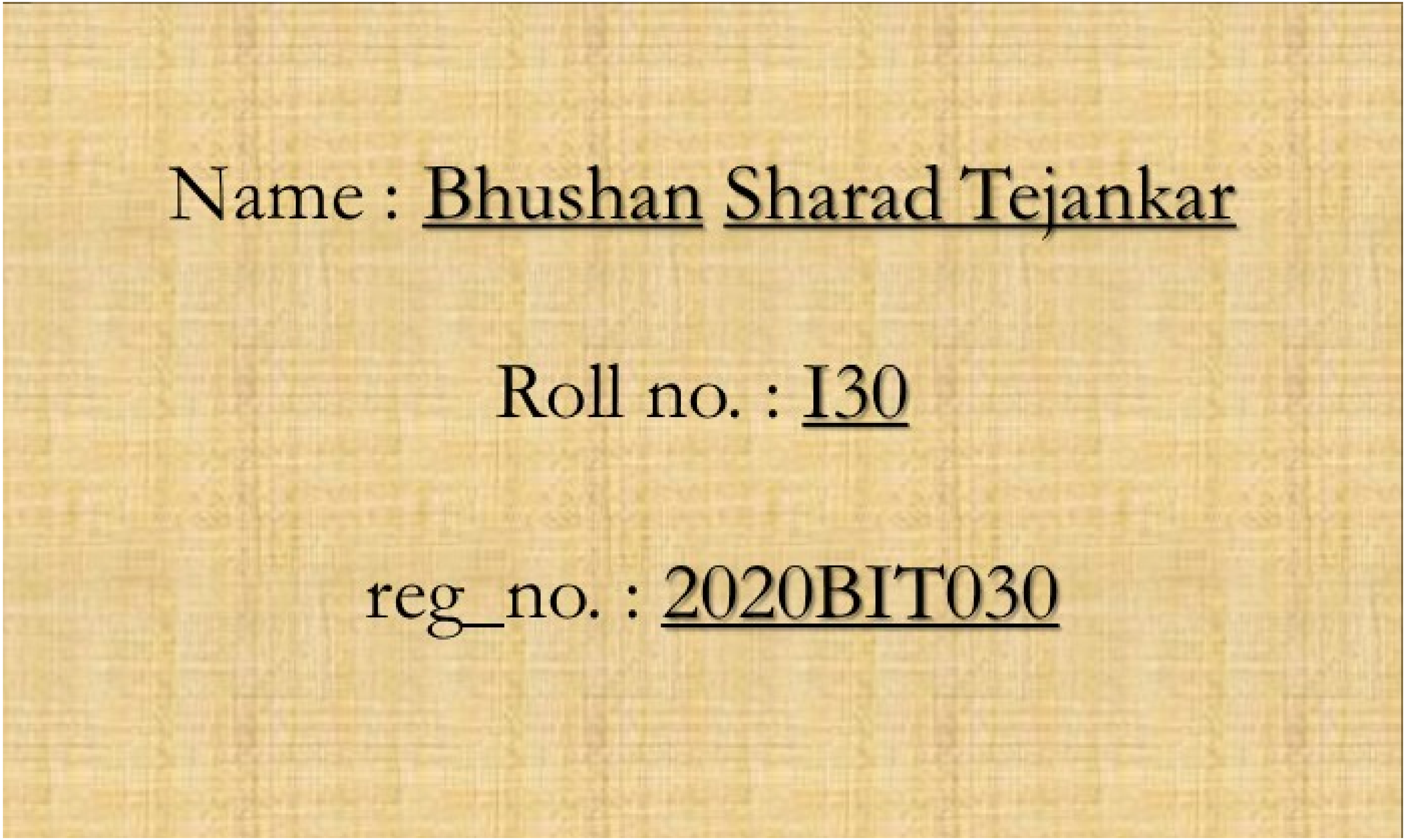
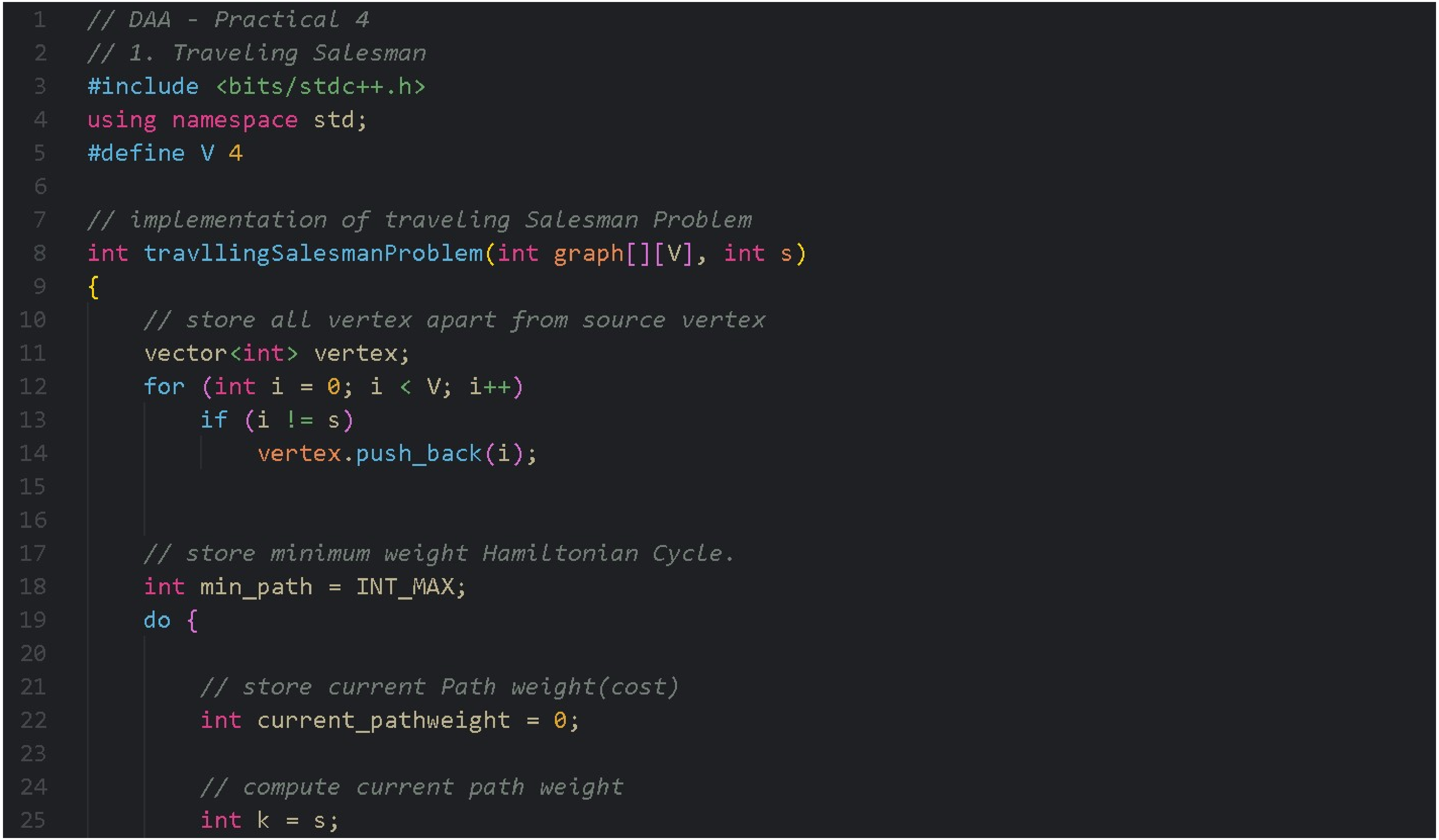


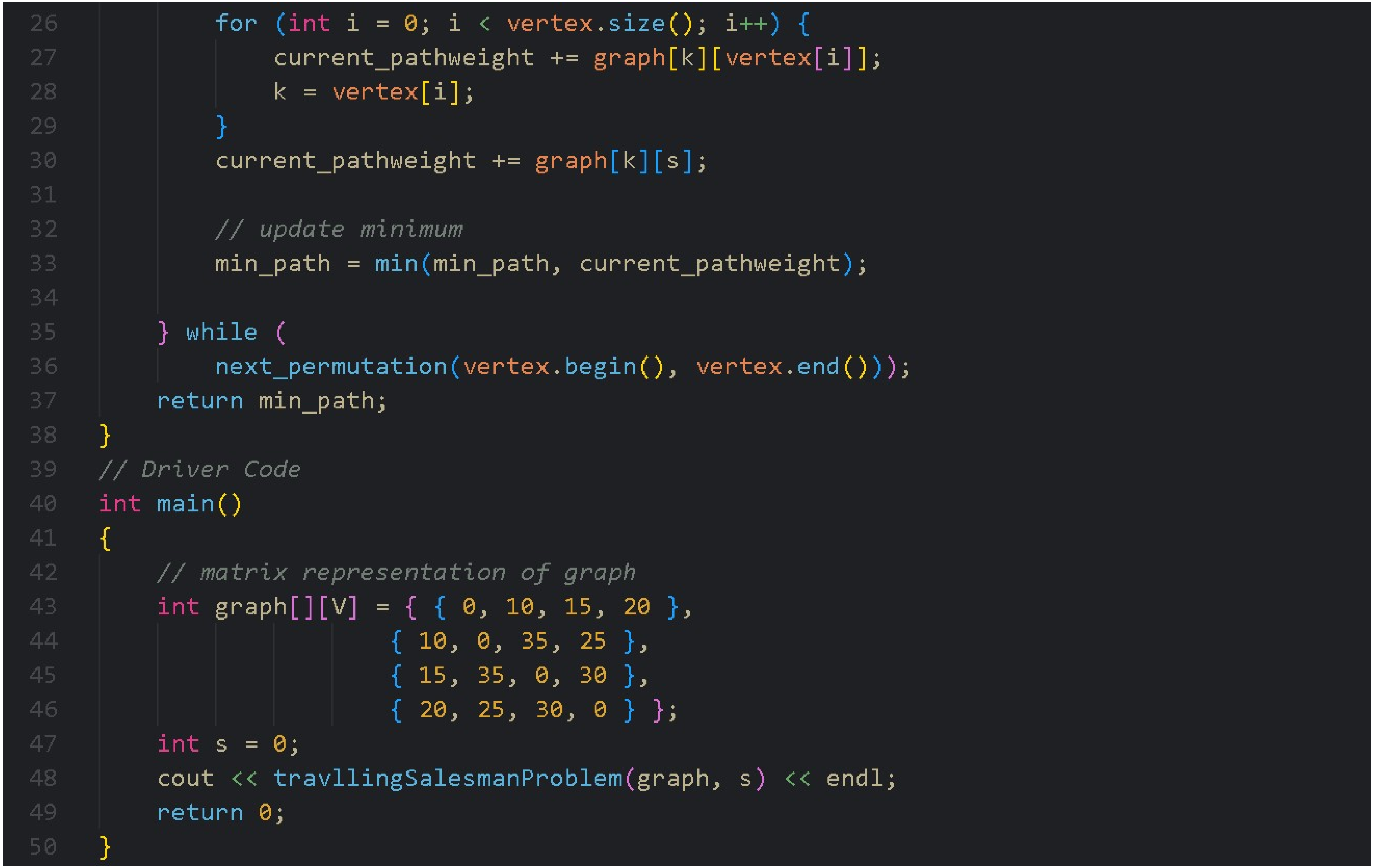
Name : Bhushan Sharad Tejankar

Roll no. : I30

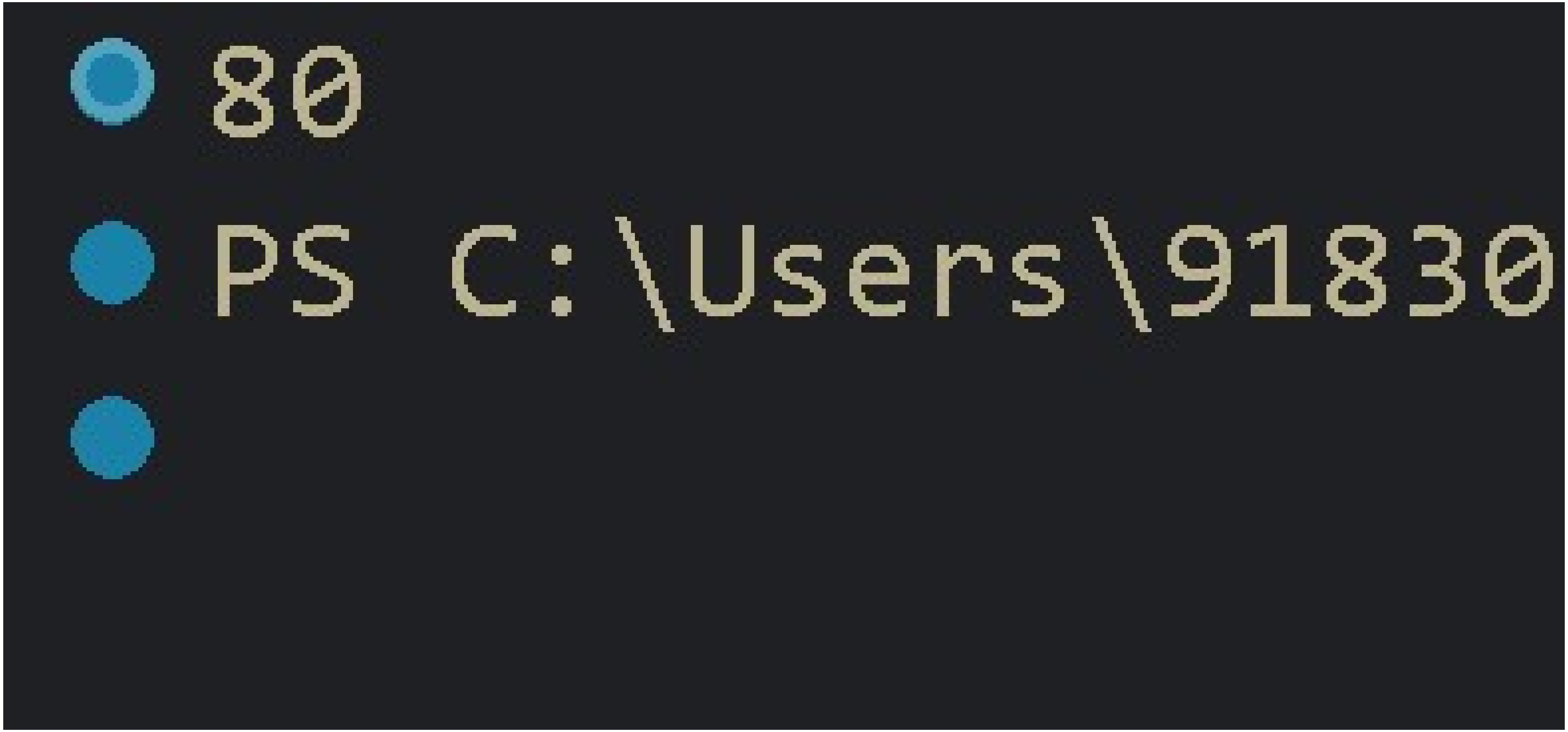
reg_no. : 2020BIT030



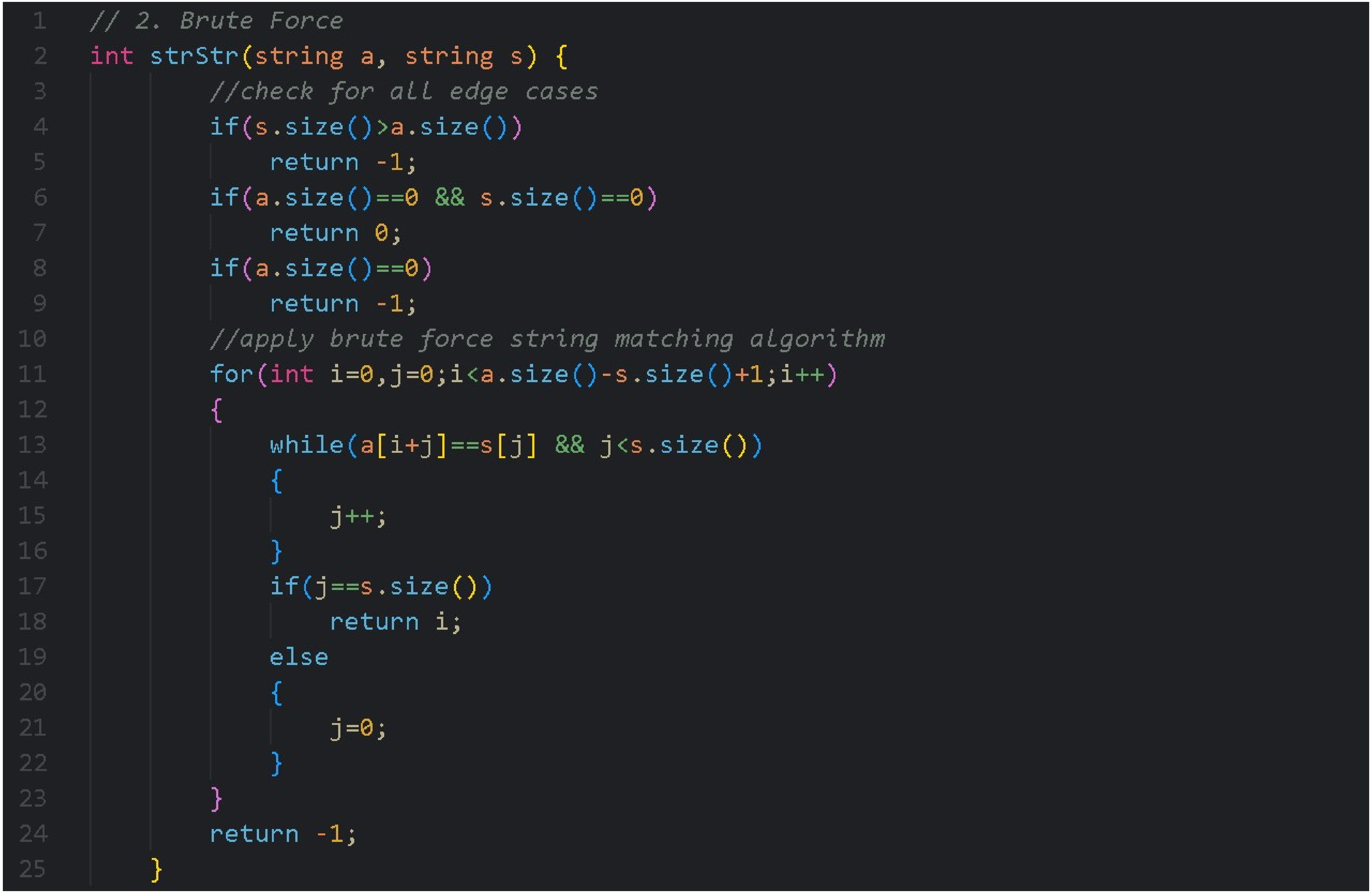
```cpp
// DAA - Practical 4
// 1. Traveling Salesman
#include <bits/stdc++.h>
using namespace std;
#define V 4

// implementation of traveling Salesman Problem
int travllingSalesmanProblem(int graph[][V], int s)
{
    // store all vertex apart from source vertex
    vector<int> vertex;
    for (int i = 0; i < V; i++)
        if (i != s)
            vertex.push_back(i);


    // store minimum weight Hamiltonian Cycle.
    int min_path = INT_MAX;
    do {

        // store current Path weight(cost)
        int current_pathweight = 0;

        // compute current path weight
        int k = s;
```



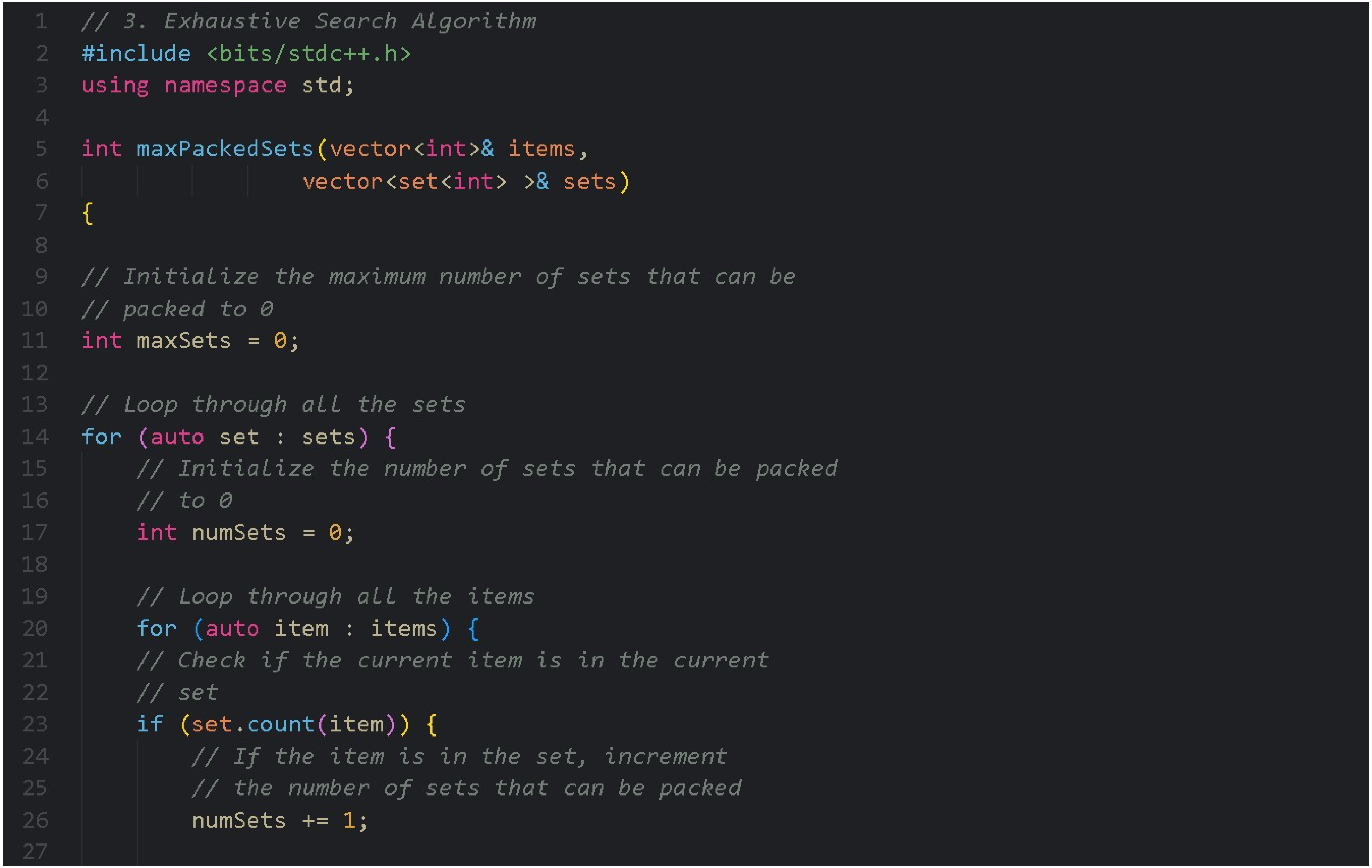
```
        for (int i = 0; i < vertex.size(); i++) {
            current_pathweight += graph[k][vertex[i]];
            k = vertex[i];
        }
        current_pathweight += graph[k][s];

        // update minimum
        min_path = min(min_path, current_pathweight);

    } while (
        next_permutation(vertex.begin(), vertex.end()));
    return min_path;
}
// Driver Code
int main()
{
    // matrix representation of graph
    int graph[][V] = { { 0, 10, 15, 20 },
                    { 10, 0, 35, 25 },
                    { 15, 35, 0, 30 },
                    { 20, 25, 30, 0 } };
    int s = 0;
    cout << travllingSalesmanProblem(graph, s) << endl;
    return 0;
}
```



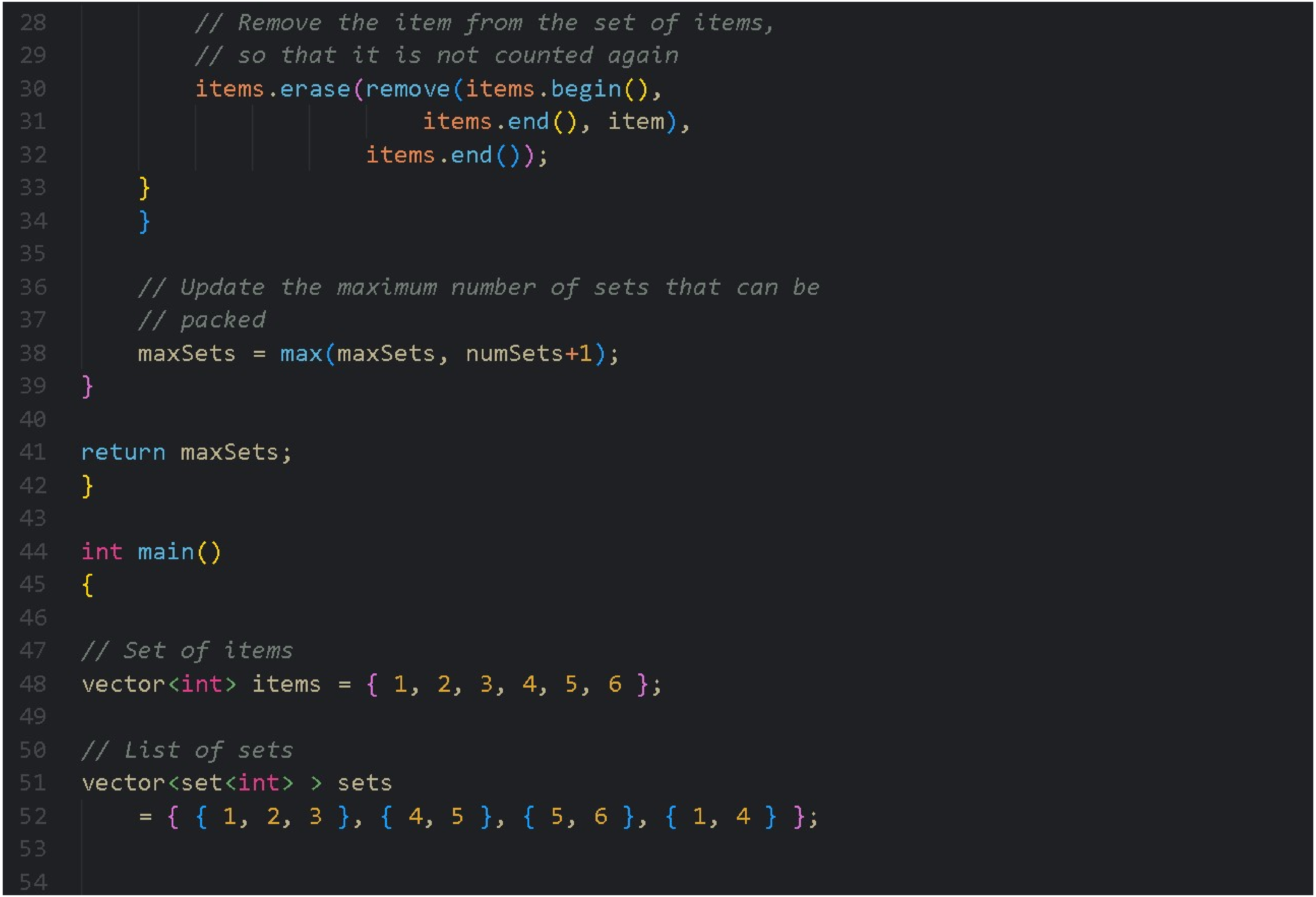
```
80
PS C:\Users\91830
```



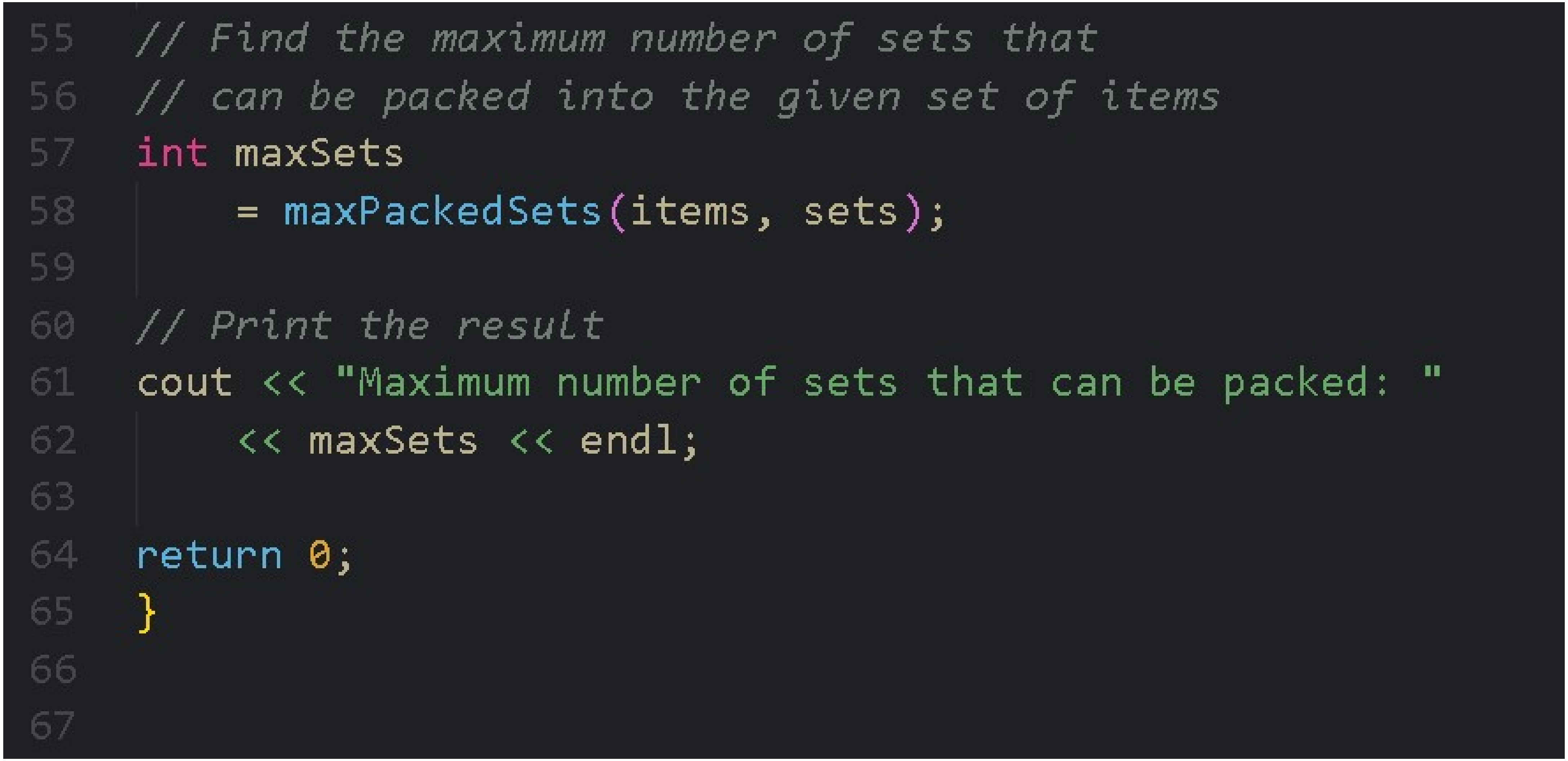
```
// 2. Brute Force
int strStr(string a, string s) {
        //check for all edge cases
        if(s.size()>a.size())
            return -1;
        if(a.size()==0 && s.size()==0)
            return 0;
        if(a.size()==0)
            return -1;
        //apply brute force string matching algorithm
        for(int i=0,j=0;i<a.size()-s.size()+1;i++)
        {
            while(a[i+j]==s[j] && j<s.size())
            {
                j++;
            }
            if(j==s.size())
                return i;
            else
            {
                j=0;
            }
        }
        return -1;
    }
```



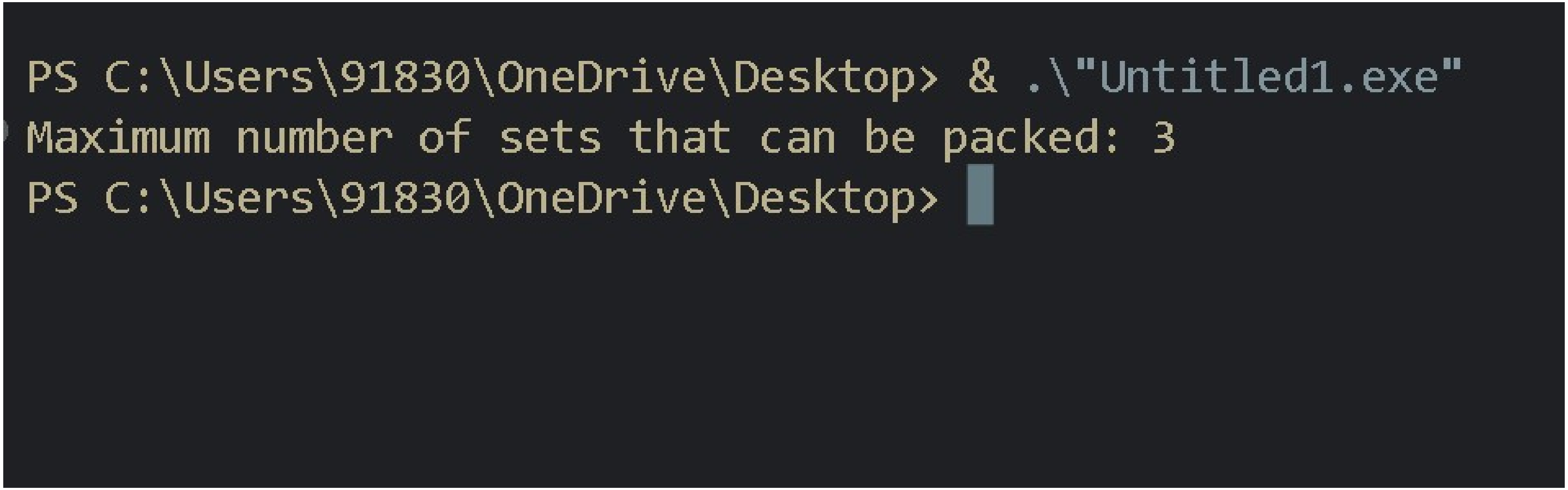
```cpp
// 3. Exhaustive Search Algorithm
#include <bits/stdc++.h>
using namespace std;

int maxPackedSets(vector<int>& items,
                vector<set<int> >& sets)
{

// Initialize the maximum number of sets that can be
// packed to 0
int maxSets = 0;

// Loop through all the sets
for (auto set : sets) {
    // Initialize the number of sets that can be packed
    // to 0
    int numSets = 0;

    // Loop through all the items
    for (auto item : items) {
    // Check if the current item is in the current
    // set
    if (set.count(item)) {
        // If the item is in the set, increment
        // the number of sets that can be packed
        numSets += 1;

```



```
    // Remove the item from the set of items,
    // so that it is not counted again
    items.erase(remove(items.begin(),
            items.end(), item),
          items.end());
  }
  }

  // Update the maximum number of sets that can be
  // packed
  maxSets = max(maxSets, numSets+1);
}

return maxSets;
}

int main()
{

// Set of items
vector<int> items = { 1, 2, 3, 4, 5, 6 };

// List of sets
vector<set<int> > sets
  = { { 1, 2, 3 }, { 4, 5 }, { 5, 6 }, { 1, 4 } };


```



```
// Find the maximum number of sets that
// can be packed into the given set of items
int maxSets
    = maxPackedSets(items, sets);

// Print the result
cout << "Maximum number of sets that can be packed: "
    << maxSets << endl;

return 0;
}
```



```
PS C:\Users\91830\OneDrive\Desktop> & .\"Untitled1.exe"
Maximum number of sets that can be packed: 3
PS C:\Users\91830\OneDrive\Desktop> 
```

THANK
YOU!