Practical No. 04

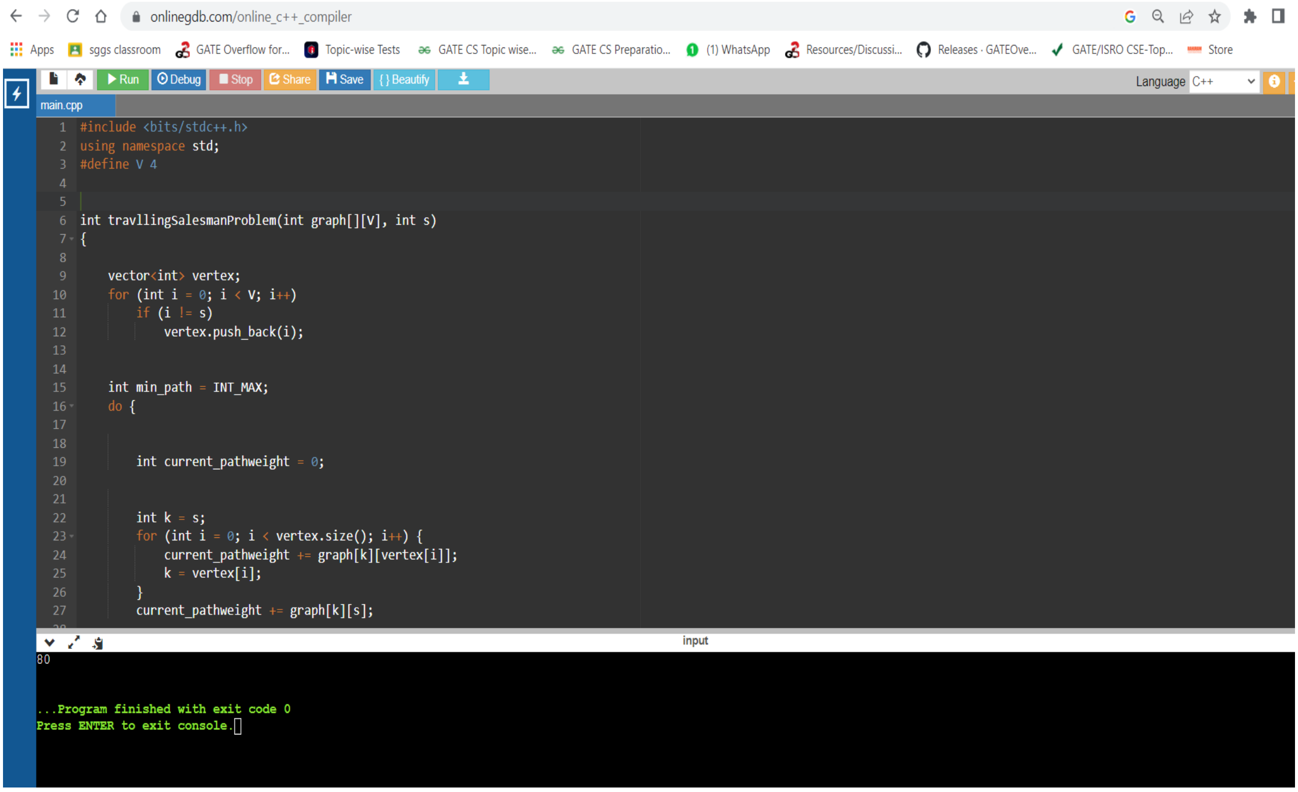
NAME: Vijiyant Tanaji Shekwalkar

REG NO: 2020BIT057

1) Travelling salesman Problem.

```
#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;

}
```



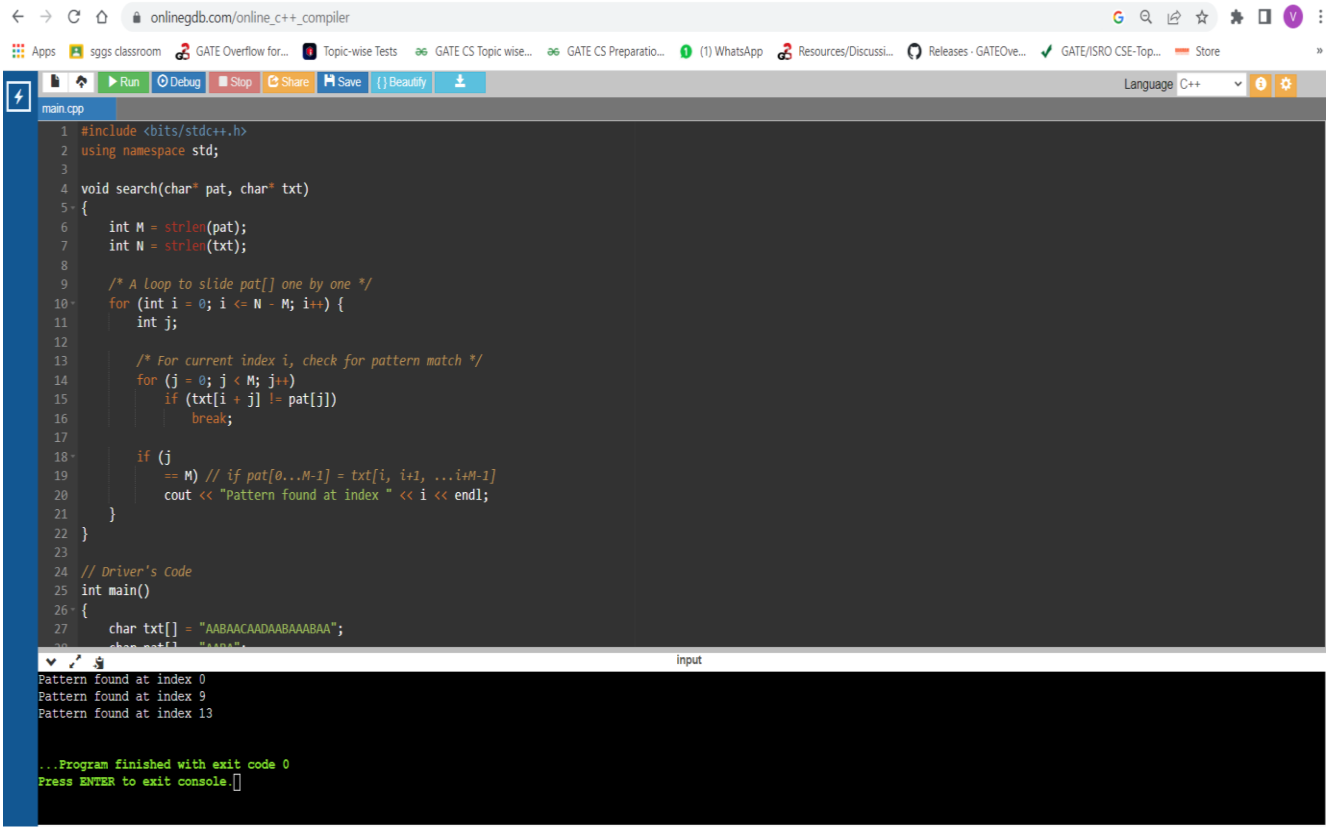
```
#include <bits/stdc++.h>
using namespace std;
#define V 4


int travllingSalesmanProblem(int graph[][V], int s)
{

    vector<int> vertex;
    for (int i = 0; i < V; i++)
        if (i != s)
            vertex.push_back(i);


    int min_path = INT_MAX;
    do {


        int current_pathweight = 0;


        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];
```

```
80


...Program finished with exit code 0
Press ENTER to exit console.
```

2) BF string Matching Algorithm

```
#include <bits/stdc++.h>
using namespace std;

void search(char* pat, char* txt)
{
   int M = strlen(pat);
   int N = strlen(txt);

   /* A loop to slide pat[] one by one */
   for (int i = 0; i <= N - M; i++) {
      int j;

      /* For current index i, check for pattern match */
      for (j = 0; j < M; j++)
         if (txt[i + j] != pat[j])
            break;

      if (j
         == M) // if pat[0...M-1] = txt[i, i+1, ...i+M-1]
         cout << "Pattern found at index " << i << endl;
   }
}

// Driver's Code
```

```
int main()
{
    char txt[] = "AABAACAADAABAAABAA";
    char pat[] = "AABA";

      // Function call
    search(pat, txt);
    return 0;
}
```



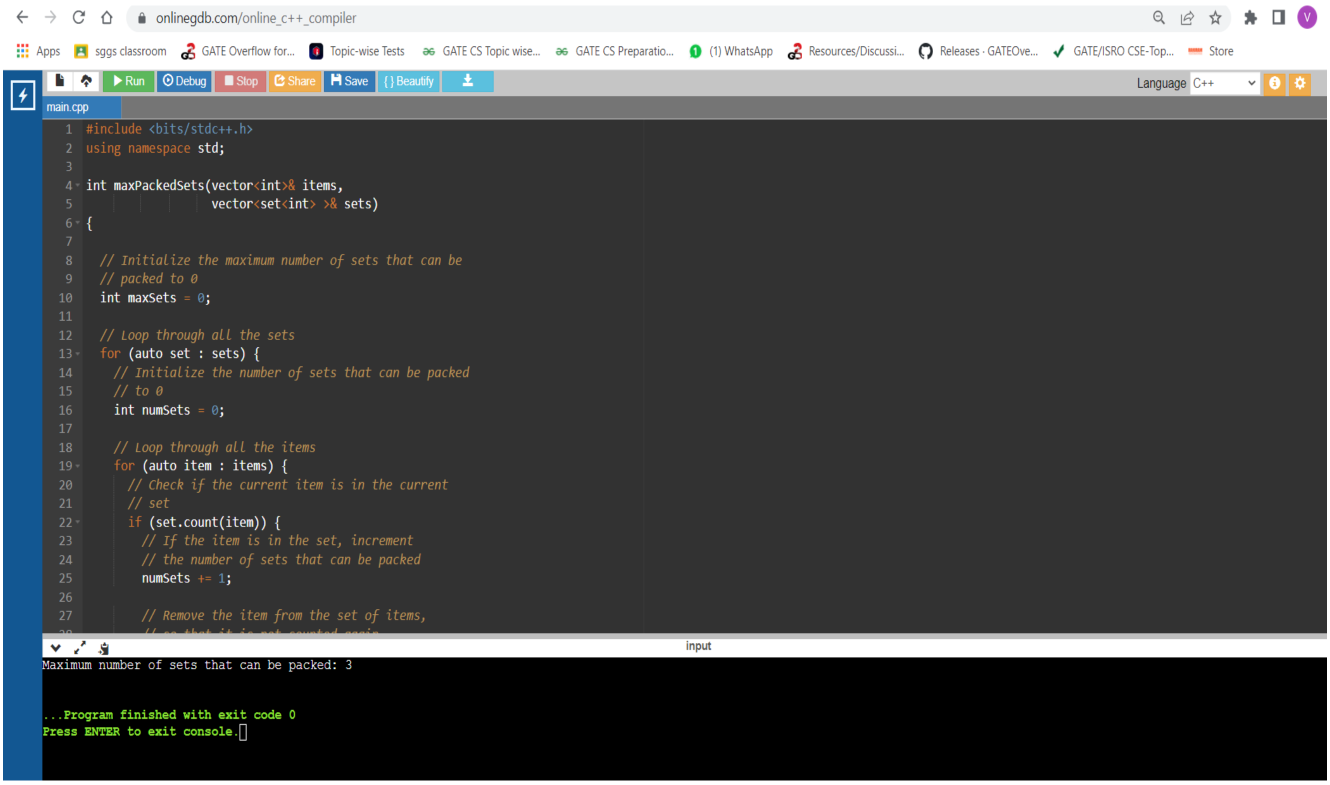
```
#include <bits/stdc++.h>
using namespace std;

void search(char* pat, char* txt)
{
    int M = strlen(pat);
    int N = strlen(txt);

    /* A loop to slide pat[] one by one */
    for (int i = 0; i <= N - M; i++) {
        int j;

        /* For current index i, check for pattern match */
        for (j = 0; j < M; j++)
            if (txt[i + j] != pat[j])
                break;

        if (j
            == M) // if pat[0...M-1] = txt[i, i+1, ...i+M-1]
            cout << "Pattern found at index " << i << endl;
    }
}

// Driver's Code
int main()
{
    char txt[] = "AABAACAADAABAAABAA";
    char pat[] = "AABA";
```

```
Pattern found at index 0
Pattern found at index 9
Pattern found at index 13


...Program finished with exit code 0
Press ENTER to exit console.
```

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;
}
```



```cpp
#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,
```

```
Maximum number of sets that can be packed: 3


...Program finished with exit code 0
Press ENTER to exit console.
```