

# 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;

}

```

The screenshot shows a web browser window with the URL `onlinegdb.com/online_c++.compiler`. The browser's address bar and tabs are visible at the top. Below the browser window is a toolbar with icons for Run, Debug, Stop, Share, Save, and other utilities. The main area displays a C++ program in a dark-themed editor. The program defines a constant `V` as 4 and implements a function `travllingSalesmanProblem`. The function takes a graph and a starting vertex `s` as input. It initializes a `vertex` vector, adds vertices from `s` to `V-1`, and then iterates through the vertices to calculate the path weight. The program finishes with an exit code of 0.

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define V 4
4
5
6 int travllingSalesmanProblem(int graph[][V], int s)
7 {
8
9     vector<int> vertex;
10    for (int i = 0; i < V; i++)
11        if (i != s)
12            vertex.push_back(i);
13
14
15    int min_path = INT_MAX;
16    do {
17
18
19        int current_pathweight = 0;
20
21
22        int k = s;
23        for (int i = 0; i < vertex.size(); i++) {
24            current_pathweight += graph[k][vertex[i]];
25            k = vertex[i];
26        }
27        current_pathweight += graph[k][s];
28
29    } while (0);
30
31    return min_path;
32 }

```

Input

```

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;
}

```

The screenshot shows an online C++ compiler interface. The code editor contains the following implementation of the search function:

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 void search(char* pat, char* txt)
5 {
6     int M = strlen(pat);
7     int N = strlen(txt);
8
9     /* A loop to slide pat[] one by one */
10    for (int i = 0; i <= N - M; i++) {
11        int j;
12
13        /* For current index i, check for pattern match */
14        for (j = 0; j < M; j++)
15            if (txt[i + j] != pat[j])
16                break;
17
18        if (j == M) // if pat[0..M-1] = txt[i, i+1, ...i+M-1]
19            cout << "Pattern found at index " << i << endl;
20    }
21 }
22
23 // Driver's Code
24 int main()
25 {
26     char txt[] = "AABAACAADAABAAABAA";
27     char pat[] = "AABA";

```

The output console shows the following results:

```

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);
```

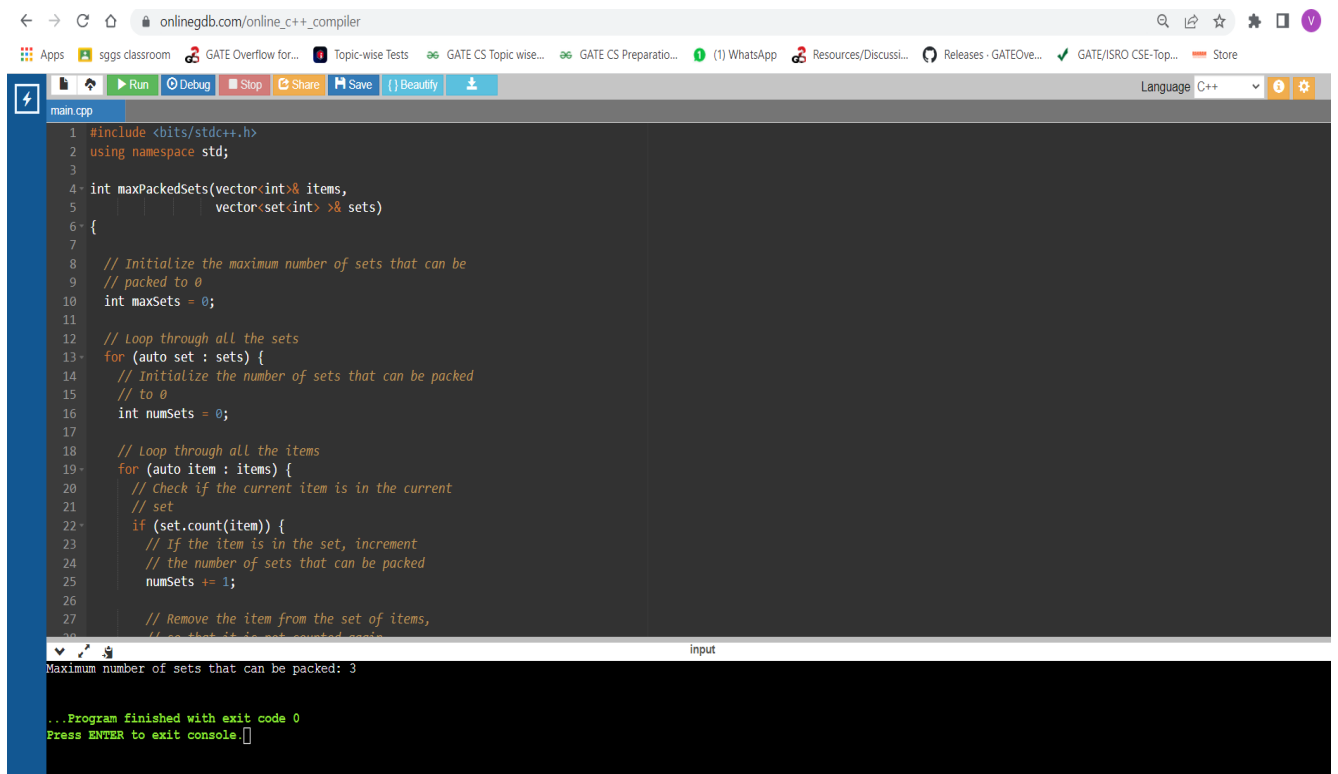
```
// Print the result
```

```
cout << "Maximum number of sets that can be packed: "
```

```
<< maxSets << endl;
```

```
return 0;
```

```
}
```



The screenshot shows a web browser window with the URL `onlinegdb.com/online_c++_compiler`. The browser's address bar and tabs are visible at the top. The main content area displays a C++ program in a dark-themed editor. The program defines a function `maxPackedSets` that takes a vector of integers and a vector of sets of integers as input. It iterates through each set and counts the number of items from the input vector that are present in the set. The maximum count across all sets is returned. The `main` function uses this function to calculate the maximum number of sets that can be packed from the given items. The output of the program is displayed in a console window at the bottom, showing the message "Maximum number of sets that can be packed: 3".

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int maxPackedSets(vector<int>& items,
5                   vector<set<int>>& sets)
6 {
7     // Initialize the maximum number of sets that can be
8     // packed to 0
9     int maxSets = 0;
10
11     // Loop through all the sets
12     for (auto set : sets) {
13         // Initialize the number of sets that can be packed
14         // to 0
15         int numSets = 0;
16
17         // Loop through all the items
18         for (auto item : items) {
19             // Check if the current item is in the current
20             // set
21             if (set.count(item)) {
22                 // If the item is in the set, increment
23                 // the number of sets that can be packed
24                 numSets += 1;
25             }
26
27             // Remove the item from the set of items,
28             // so that it is not counted again
29         }
30
31         // Update the maximum number of sets that can be packed
32         maxSets = max(maxSets, numSets);
33     }
34
35     return maxSets;
36 }
37
38 int main() {
39     vector<int> items = { 1, 2, 3, 4, 5, 6 };
40     vector<set<int>> sets = { { 1, 2, 3 }, { 4, 5 }, { 5, 6 }, { 1, 4 } };
41
42     int result = maxPackedSets(items, sets);
43
44     cout << "Maximum number of sets that can be packed: "
45           << result << endl;
46
47     return 0;
48 }
```

Maximum number of sets that can be packed: 3

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