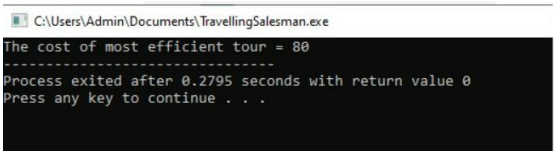
Name-Bhakti Bapurao patil

Roll_no-2020BIT064

1) Travelling salesman problem

```
#include <iostream>
using namespace std;
const int n = 4;
const int MAX = 1000000;
int dist[n + 1][n + 1] = {
        { 0, 0, 0, 0, 0 }, { 0, 0, 10, 15, 20 },
        { 0, 10, 0, 25, 25 }, { 0, 15, 25, 0, 30 },
        { 0, 20, 25, 30, 0 },
};

int memo[n + 1][1 << (n + 1)];

int fun(int i, int mask)
{

        if (mask == ((1 << i) | 3))
                return dist[1][i];

        if (memo[i][mask] != 0)
                return memo[i][mask];

        int res = MAX;
        for (int j = 1; j <= n; j++)
                if ((mask & (1 << j)) && j != i && j != 1)
                        res = std::min(res, fun(j, mask &
(~(1 << i)))

                + dist[j][i]);
```

```
        return memo[i][mask] = res;
}

int main()
{
        int ans = MAX;
        for (int i = 1; i <= n; i++)
                ans = std::min(ans, fun(i, (1 << (n + 1)) - 1)

        + dist[i][1]);

        printf("The cost of most efficient tour = %d",
ans);

        return 0;
}
```

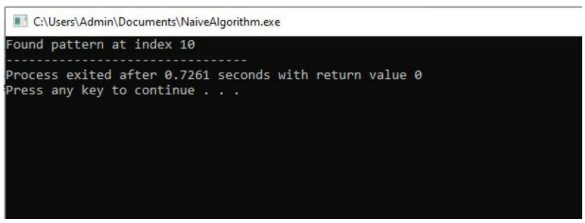


2) BF string matching algorithm

```
#include <bits/stdc++.h>
void computeLPSArray(char* pat, int M, int* lps);
void KMPSearch(char* pat, char* txt)
{
        int M = strlen(pat);
        int N = strlen(txt);
        int lps[M];
        computeLPSArray(pat, M, lps);

        int i = 0;
        int j = 0;
        while ((N - i) >= (M - j)) {
```

```
		if (pat[j] == txt[i]) {
			j++;
			i++;
		}

		if (j == M) {
			printf("Found pattern at index %d
", i - j);
			j = lps[j - 1];
		}

		else if (i < N && pat[j] != txt[i]) {
			if (j != 0)
				j = lps[j - 1];
			else
				i = i + 1;
		}
	}
}

void computeLPSArray(char* pat, int M, int* lps)
{
	int len = 0;

	lps[0] = 0;

	int i = 1;
	while (i < M) {
		if (pat[i] == pat[len]) {
			len++;
			lps[i] = len;
			i++;
		}
		else
		{
```

```
                        if (len != 0) {
                                len = lps[len - 1];

                        }
                        else
                        {
                                lps[i] = 0;
                                i++;
                        }
                }
        }
}
int main()
{
        char txt[] = "ABABDABACDABABCABAB";
        char pat[] = "ABABCABAB";
        KMPSearch(pat, txt);
        return 0;
}
```



3) Exhaustive search

```
#include <bits/stdc++.h>
using namespace std;
int maxPackedSets(vector<int>& items,
                                vector<set<int> >& sets)
{
int maxSets = 0;

for (auto set : sets) {
```

```
        int numSets = 0;
        for (auto item : items) {
        if (set.count(item)) {
                numSets += 1;
                items.erase(remove(items.begin(),
                                                items.end(),
item),
                                        items.end());
        }
        }
        maxSets = max(maxSets, numSets+1);
}

return maxSets;
}

int main()
{
vector<int> items = { 1, 2, 3, 4, 5, 6 };

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

int maxSets
        = maxPackedSets(items, sets);

cout << "Maximum number of sets that can be packed: "
        << maxSets << endl;

return 0;
}
```

C:\Users\Admin\Documents\ExhaustiveSearch.exe

```
Maximum number of sets that can be packed: 3
--------------------------------
Process exited after 0.1651 seconds with return value 0
Press any key to continue . . .
```