1] Travelling salesman problem

```cpp
#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;
}
```

```
OBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

C:\Users\HP\Desktop\DSA> cd "c:\Users\HP\Desktop\DSA\daa_pr\" ; if ($?) { g
e cost of most efficient tour = 80
C:\Users\HP\Desktop\DSA\daa_pr>
```

2) BF string Matching Algorithm

DAA_PRACTICAL > string_matching.cpp > main()

```cpp
#include <iostream>
#include <string>
using namespace std;
int BF(string text, string pattern) {
  int n = text.length();
  int m = pattern.length();
  for (int i = 0; i <= n - m; i++) {
    int j = 0;
    while (j < m && text[i + j] == pattern[j]) {
      j++;
    }
    if (j == m) {
      return i;
    }
  }
  return -1;
}
int main() {
  string text = "shubham";
  string pattern = "shu";
  int pos = BF(text, pattern);
  if (pos != -1) {
    cout << "Pattern found at position: " << pos <<endl;
  } else {
    cout << "Pattern not found" <<endl;
  }
  return 0;
}
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
Pattern found at position: 0
PS D:\DSA PRACTICE\DAA_PRACTICAL>
```

3) Exhaustive Search Algorithm

DAA_PRACTICAL > Exhaustive.cpp > ...

```cpp
#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
```

```cpp
int maxSets
    = maxPackedSets(items, sets);

cout << "Maximum number of sets that can be packed: "
    << maxSets << endl;
return 0;
}
```

PROBLEMS 18 OUTPUT TERMINAL DEBUG CONSOLE

```
PS D:\DSA PRACTICE> cd "d:\DSA PRACTICE\DAA_PRACTICAL\" ; if ($?) { g++ saleman_daa.
The cost of most efficient tour = 80
```