LAB 8

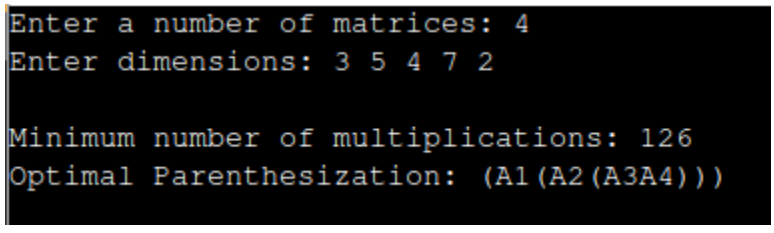
Matrix chain Multiplication using DP :

PROGRAM :

```
#include <iostream>
#include <climits>
using namespace std;

void printOptimalParens(int i, int j, int **K){
    if(i == j){
        cout << "A" << i;
        return;
    }

    cout << "(";
    printOptimalParens(i, K[i][j], K);
    printOptimalParens(K[i][j] + 1, j, K);
    cout << ")";
}

int main(){

    int n;
    cout << "Enter a number of matrices: ";
    cin >> n;

    int *P = new int[n+1];

    cout << "Enter dimensions: ";
    for(int i=0; i<n+1; i++)
        cin >> P[i];

    int **C = new int*[n+1];
    int **K = new int*[n+1];

    for(int i=0; i<n+1; i++){
        C[i] = new int[n+1];
        K[i] = new int[n+1];
    }
```

```
    for(int i=1; i<n+1; i++)
        C[i][i] = 0;


    for(int l=2; l<n+1; l++){
        for(int i=1; i<=n-l+1; i++){
            int j = i+l-1;
            C[i][j] = INT_MAX;

            for(int k=i; k<j; k++){
                int q = C[i][k] + C[k+1][j] + P[i-1]*P[k]*P[j];

                if(q < C[i][j]){
                    C[i][j] = q;
                    K[i][j] = k;
                }
            }

        }
    }


    cout << "\nMinimum number of multiplications: " << C[1][n] << endl;
    cout << "Optimal Parenthesization: ";
    printOptimalParens(1, n, K);
    cout << endl;


    return 0;
}
```

OUTPUT :



Edit distance problem using DP :

PROGRAM :

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;

int minEditDist(string s1, string s2){

    int m = s1.length();
    int n = s2.length();

    vector<vector<int>> dp(m+1, vector<int>(n+1));

    for(int i=0; i<=m; i++)
        dp[i][0] = i;

    for(int j=0; j<=n; j++)
        dp[0][j] = j;

    for(int i=1; i<=m; i++){
        for(int j=1; j<=n; j++){
            if(s1[i-1] == s2[j-1])
                dp[i][j] = dp[i-1][j-1];
            else{
                dp[i][j] = 1+ min({dp[i][j-1], dp[i-1][j], dp[i-1][j-1]});

            }
        }
    }

    return dp[m][n];
}

int main(){

    string s1, s2;

    cout << "Enter a first string: ";
    cin >> s1;

    cout << "Enter a second string: ";
```

```
            cin >> s2;

            int result = minEditDist(s1, s2);

            cout << "Minimum Edit Distance = " << result << endl;


            return 0;
        } (T[i - 1][j] <= 1 + T[i][j - d[i - 1]]) {
            T[i][j] = T[i - 1][j];
        } else {
            T[i][j] = 1 + T[i][j - d[i - 1]];
            sol[i][j] = i;
        }
      } else {
        T[i][j] = T[i - 1][j];
      }
    }
  }

  return (T[n][N] == INT_MAX) ? -1 : T[n][N];
}

void printSolution(vector<vector<int>>& sol, vector<int>& d, int N, int n) {
  cout << "\nCoins selected (index, value):\n";

  int i = n, j = N;
  while (j > 0 && i > 0) {
    if (sol[i][j] == i) {
      cout << "Coin " << i << " (Value: " << d[i - 1] << ")\n";
      j -= d[i - 1];
    } else {
      i--;
    }
  }
}

int main() {
  vector<int> denominations = {1, 4, 6};
  int N = 8;

  vector<vector<int>> T, sol;
  int result = minCoins(N, denominations, T, sol);

  cout << "DP Table:\n";
```

```
    for (auto& row : T) {
        for (int val : row)
            cout << (val == INT_MAX ? 0 : val) << " ";
        cout << endl;
    }

    if (result != -1) {
        printSolution(sol, denominations, N, denominations.size());
    cout << "\nMinimum coins required = " << result << endl; }
    else {
        cout << "\nSolution not possible\n";
    }

    return 0;
}
```

OUTPUT :