```
#include<iostream>

using namespace std;


int multMatrix(int *vec){
  int size = 7;
  int matrix[size][size];

  for (int i = 1; i < size; ++i)
        matrix[i][i] = 0;

   for (int h = 2; h < size; ++h){
        for (int i = 1; i < size - h + 1; i++){
            int j = i + h - 1;
            matrix[i][j] = 99999;
            for (int k = i; k < j; k++){
                int temp = matrix[i][k] + matrix[k + 1][j] + vec[i - 1] * vec[k] * vec[j];
                if (temp < matrix[i][j])
                    matrix[i][j] = temp;
            }
        }
    }
    return matrix[1][size - 1];

}


int main(){


    int vec[7] = {30, 35, 15, 5, 10, 20, 25};

    cout << multMatrix(vec);
}
```

explicación:

pasos para resolver el problema:

1. crear una matriz de nxn
2. llenar la matriz con valores infinitos
3. llenar la diagonal con 0
4. llenar la matriz con los valores de la matriz de la multiplicacion de matrices
5. retornar el valor de la matriz en la posicion [1][size-1]

complejidad temporal: O(n^3)

capturas de pantalla: