**Brackets in Matrix Chain Multiplication**

Given a sequence of matrices, find the most efficient way to multiply these matrices together. The problem is not actually to perform the multiplications, but merely to decide in which order to perform the multiplications. There are many options to multiply a chain of matrices because matrix multiplication is associative i.e. no matter how one parenthesize the product, the result will be the same.

Example:  
 if you had four matrices A, B, C, and D, you would have:

(ABC)D = (AB)(CD) = A(BCD) = ....

However, the order in which one parenthesize the product affects the number of simple arithmetic operations needed to compute the product, or the efficiency.

For example:

A: 10 × 30 matrix

B : 30 × 5 matrix

C : 5 × 60 matrix

Then,

     (AB)C = (10×30×5) + (10×5×60)

= 1500 + 3000

= 4500 operations

      A(BC) = (30×5×60) + (10×30×60)

= 9000 + 18000

= 27000 operations.

Given an array **arr[]** which represents the chain of matrices such that the ith matrix Ai is of dimension **arr[i-1] x arr[i]**.  
Your task is to write a program that should print the optimal way to multiply the matrix chain such that minimum number of multiplications operations are needed to multiply the chain. Represent first matrix as starting Alphabet of the english dictionary i.e. 'A', and the rest so on.

**Input: p[] = {40, 20, 30, 10, 30}**

**Output: Optimal parenthesization is ((A(BC))D)**

There are 4 matrices of dimensions 40x20,

20x30, 30x10 and 10x30. Let the input 4

matrices be A, B, C and D. The minimum

number of multiplications are obtained

by putting parenthesis in following way

(A(BC))D --> 20\*30\*10 + 40\*20\*10 + 40\*10\*30

**Input: p[] = {10, 20, 30, 40, 30}**

**Output: Optimal parenthesization is (((AB)C)D)**

There are 4 matrices of dimensions 10x20,

20x30, 30x40 and 40x30. Let the input 4

matrices be A, B, C and D. The minimum

number of multiplications are obtained by

putting parenthesis in following way

((AB)C)D --> 10\*20\*30 + 10\*30\*40 + 10\*40\*30

**Input:**  
The first line of the input contains an integer **T**, denoting the number of test cases. Then **T**test case follows. The first line of each test case contains an integer **N**, denoting the number of elements in the array.  
Then next line contains **N**space separated integers denoting the values of the element in the array.  
  
**Output:**  
For each test case the print the minimum number of operations needed to multiply the chain.

**Constraints:**  
1<=T<=100  
2<=N<=27  
1<=A[]<=500  
  
**Example:  
Input:**  
2  
5  
1 2 3 4 5  
3  
3 3 3  
**Output:**  
(((AB)C)D)  
(AB)

* GFG SOLUTION :

// in matrix chain multiplication.

#include<bits/stdc++.h>

using namespace std;

// Function for printing the optimal

// parenthesization of a matrix chain product

void printParenthesis(int i, int j, int n,

                      int \*bracket, char &name)

{

    // If only one matrix left in current segment

    if (i == j)

    {

        cout << name++;

        return;

    }

    cout << "(";

    // Recursively put brackets around subexpression

    // from i to bracket[i][j].

    // Note that "\*((bracket+i\*n)+j)" is similar to

    // bracket[i][j]

    printParenthesis(i, \*((bracket+i\*n)+j), n,

                     bracket, name);

    // Recursively put brackets around subexpression

    // from bracket[i][j] + 1 to j.

    printParenthesis(\*((bracket+i\*n)+j) + 1, j,

                     n, bracket, name);

    cout << ")";

}

// Matrix Ai has dimension p[i-1] x p[i] for i = 1..n

// Please refer below article for details of this

// function

// <https://goo.gl/k6EYKj>

void matrixChainOrder(int p[], int n)

{

    /\* For simplicity of the program, one extra

       row and one extra column are allocated in

        m[][]. 0th row and 0th column of m[][]

        are not used \*/

    int m[n][n];

    // bracket[i][j] stores optimal break point in

    // subexpression from i to j.

    int bracket[n][n];

    /\* m[i,j] = Minimum number of scalar multiplications

    needed to compute the matrix A[i]A[i+1]...A[j] =

    A[i..j] where dimension of A[i] is p[i-1] x p[i] \*/

    // cost is zero when multiplying one matrix.

    for (int i=1; i<n; i++)

        m[i][i] = 0;

    // L is chain length.

    for (int L=2; L<n; L++)

    {

        for (int i=1; i<n-L+1; i++)

        {

            int j = i+L-1;

            m[i][j] = INT\_MAX;

            for (int k=i; k<=j-1; k++)

            {

                // q = cost/scalar multiplications

                int q = m[i][k] + m[k+1][j] + p[i-1]\*p[k]\*p[j];

                if (q < m[i][j])

                {

                    m[i][j] = q;

                    // Each entry bracket[i,j]=k shows

                    // where to split the product arr

                    // i,i+1....j for the minimum cost.

                    bracket[i][j] = k;

                }

            }

        }

    }

    // The first matrix is printed as 'A', next as 'B',

    // and so on

    char name = 'A';

    cout << "Optimal Parenthesization is : ";

    printParenthesis(1, n-1, n, (int \*)bracket, name);

    cout << "nOptimal Cost is : " << m[1][n-1];

}

// Driver code

int main()

{

    int arr[] = {40, 20, 30, 10, 30};

    int n = sizeof(arr)/sizeof(arr[0]);

    matrixChainOrder(arr, n);

    return 0;

}

* MY SOLUTION:

#include<bits/stdc++.h>

#define int long long int

using namespace std;

int n;

int dp[28][28],bracket[28][28];

char c;

void print(int i,int j)

{

if(i>=j)

{

cout<<c++;

return;

}

cout<<"(";

print(i,bracket[i][j]);

print(bracket[i][j]+1,j);

cout<<")";

}

int solve(int a[],int i,int j)

{

if(i>=j)

return 0;

if(dp[i][j]!=-1)

return dp[i][j];

int mnans=1e18;

for(int k=i;k<j;k++)

{

int tempans=solve(a,i,k)+solve(a,k+1,j)+a[i-1]\*a[k]\*a[j];

if(mnans>tempans)

{

mnans=tempans;

bracket[i][j]=k;

}

}

return dp[i][j]=mnans;

}

main()

{

int t;

cin>>t;

while(t--)

{

memset(dp,-1,sizeof(dp));

memset(bracket,-1,sizeof(bracket));

cin>>n;

int a[n],i;

for(i=0;i<n;i++)

cin>>a[i];

int x=solve(a,1,n-1);

c='A';

print(1,n-1);

cout<<"\n";

}

}