Jugando con números

https://jutge.org/problems/P26716_es

Dados n números, calculad qué resultados distintos es posible obtener con los operadores suma y producto, poniendo tantos paréntesis como se quiera, pero sin cambiar el orden de los operandos. Por ejemplo, con la secuencia 2, 1, 3 únicamente es posible obtener 5, 6, 8 y 9. Éstas son algunas de las maneras posibles:

$$\begin{aligned} 5 &= (2 \cdot 1) + 3, \\ 6 &= 2 \cdot (1 \cdot 3), \\ 8 &= 2 \cdot (1 + 3), \\ 9 &= (2 + 1) \cdot 3. \end{aligned}$$

Entrada

La entrada contiene un número arbitrario (pero no superior a 100) de casos, cada uno de los cuales con $1 \leq n \leq 9$, seguido de n números entre 1 y 9.

Salida

Para cada caso, escribid todos los posibles resultados en orden creciente. Separad los resultados consecutivos con comas. Escribid exactamente 10 números por línea (excepto, tal vez, la última). Acabad el listado con un punto. Fijaos en los ejemplos.

Solución

Para este problema vamos a usar la estructura **set** de c++, estructuras similares se pueden encontrar en otros lenguajes. La idea de esta estructura de datos es que inserir un elemento, en este caso enteros, tiene coste $O(logn)$, donde n es el número de elementos del set en este momento. Además solo se añade un elemento si no existe ya en el set. Otra propiedad interesante es que al recorrer los elementos del set en un for estos están ordenados de menor a mayor.

Supongamos que tenemos n números, cualquier combinazión de operaciones y paréntesis puede ser ampliado añadiendo el máximo número de paréntesis sin que se canvie el orden de las operaciones. Por ejemplo:

$$\begin{aligned} 2 \cdot 1 + 4 \cdot 2 &= ((2 \cdot 1) + (4 \cdot 2)) \\ 3 + 2 + 1 \cdot 6 &= (3 + (2 + (1 \cdot 6))) \\ (2 + 4) \cdot 4 + 3 &= (((2 + 4) \cdot 4) + 3) \end{aligned}$$

Entonces podemos suponer que todas las operaciones están paranterizadas al máximo. En este caso, si R_{ij} es el conjunto de números que se pueden obtener con los enteros entre la posición i y la j de los n originales tenemos que:

$$R_{ij} = \bigcup_{k=i+1}^{j} \{(R_{i(k-1)} + R_{kj}), (R_{i(k-1)} \cdot R_{kj})\}$$

Es decir, podemos conseguir el valor de R_ij mirando todas las divisiones de este intervalo en dos grupos y operando los posibles resultados de estos subintervalos entre ellos.

Esto se puede hacer con una función recursiva, es importante guardarse los valores ya calculados de R_ij para no repetir computaciones innecesarias. Esta técnica de guardar resultados es conocida como programación dinámica.

Código

C++

```cpp
#include <bits/stdc++.h>

using namespace std;

using si = set<int>;
using vsi = vector<si>;
using vvsi = vector<vsi>;

vvsi D;

// Funcion recursiva que calcula los posibles
// valores de cada intervalo
void calcula(int l, int r) {
  // Si ya lo hemos calculado
  // no lo recalculamos
  if (D[l][r].size()) return;

  // Miramos todas las opciones de parenterizar
  for (int i = l; i < r; ++i) {
    // Primero calculamos los dos intervalos
    // en los que separamos
    calcula(l, i);
    calcula(i+1, r);

    // Una vez tenemos los posibles valores
    // de cada intervalo los fusionamos
    for (int x : D[l][i]) {
      for (int y : D[i+1][r]) {
        D[l][r].insert(x + y);
        D[l][r].insert(x * y);
      }
    }
  }
}

int main() {
  int n;
  while (cin >> n) {
    // Creamos una matriz de sets de n x n
    D = vvsi(n, vsi(n));

    // Inicializamos los intervalos de un solo
    // entero con el unico valor que pueden tener
    for (int i = 0; i < n; ++i) {
```

```
45        int x;
46        cin >> x;
47        D[i][i].insert(x);
48      }
49
50      // Llamamos la funcion para el intervalo total
51      calcula(0, n-1);
52
53      // Escribimos el resultado teniendo en cuenta
54      // el formato que se nos pide
55      int contador = 0;
56      for (int x : D[0][n-1]) {
57        if (contador)
58          cout << ',';
59
60        if (contador and contador%10 == 0)
61          cout << endl;
62
63        cout << x;
64        contador++;
65      }
66
67      cout << '.' << endl;
68    }
69  }
```