URJC Solves it

Helados de Cucurucho - 325 Acepta El Reto

Descripción del problema

En una familia nadie se pone de acuerdo para decidir el orden en el que se comen los helados en un cucurucho.

Nos dan indicios de que el problema está ambientado en formas de colocar bolas de chocolates y de vainillas en un cucurucho

Entrada

T casos de prueba.

Cada caso tiene el número de bolas de vainilla (V) y el número de bolas de chocolate (C).

La descripción del problema garantiza que:

- C+V <= 15
- C+V > 0.

Salida

Por cada caso, se debe escribir la lista de posibles configuraciones de helados de Chocolate y Vainilla en **orden lexicográfico** tal que:

En cualquier string generado haya tantas bolas de vainilla y de chocolate como te han pedido en la entrada

Ejemplos

Entrada: 1 1 (1C y 1V)

Salida: CV VC

Entrada: 2 1 (2C y 1V)

Salida: CCV CVC VCC

Razonamiento

Pensar en que tenemos un ‘pool’ de bolas de helado que podemos colocar en el orden que queramos.

Sin embargo la salida debe ser **lexicográfica**.

El orden lexicográfico es una relación de orden definida sobre el producto cartesiano de conjuntos ordenados. Por defecto, ¡Los lenguajes de programación que utilizamos lo tienen incluido!

Comparación de strings en C++

En el ejemplo, PEPE es mayor a PEPA lexicográficamente

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

using namespace std;

int main(){
  string A = "PEPE";
  string B = "PEPA";
  if(A < B){
    printf("%s es lexicograficamente menor que %s\n", A.c_str(), B.c_str());
  }
  else{
    printf("%s es lexicograficamente mayor que %s\n", A.c_str(), B.c_str());
  }
  return 0;
}
```

Comparación de strings en Java

En el ejemplo, PEPE es mayor a PEPA lexicográficamente

```
1       String a = "PEPI";
2       String b = "PEPE";
3       if(a.compareTo(b)<0)
4           System.out.println(a+" es lexicográficamente"
5                   + " menor que "+b);
6       else
7           System.out.println(b+" es lexicográficamente"
8                   + " mayor o igual que "+b);
```

Pool de helados

Dándonos cuenta de cómo se ordena lexicográficamente una lista de strings… ¿Vale la pena?

Tenemos un pool de helados y sabemos cuales necesitamos de cada uno, por lo que **podemos armar la lista de manera lexicográfica** desde el principio en lugar de guardar todos los posibles resultados.

Generar helados

Si tenemos un conjunto de bolas de Chocolate (C) y Vainilla (V) podemos generar un string de tamaño C+V con todos los chocolates al principio y todas las vainillas al final.

C=2, V=1

String inicial → CCV

Generar helados

Este sería el lexicográficamente menor a todos, el siguiente sería desplazar 1V hacia la derecha, es decir:

C=2, V=1

String → CVC

Generar helados

Si seguimos esta idea, nos daremos cuenta que existe una relación entre todos los strings, y que podemos asociarlo a una **recursión.**

La recursión llevaría cuenta de cuantas bolas de helado nos quedan (tanto de vainilla como de chocolate) y qué string llevamos hasta el momento.

Si C=0 y V=0, imprimimos el string que tengamos hasta ese momento.

Generar helados

```
 5 void rec(int choco_left, int vaini_left, string res){
 6   if(choco_left == 0 && vaini_left == 0) printf("%s\n",res.c_str()); // Imprimir
 7   else{
 8     if(choco_left > 0)
 9       rec(choco_left-1, vaini_left, res + "C"); // Poner Choco
10     if(vaini_left > 0)
11       rec(choco_left, vaini_left-1, res + "V"); // Poner Vainilla
12   }
13 }
```

Generar helados (Java)

```java
private static void rec(int choco_left, int vainilla_left,
        StringBuilder sb) {
    if(choco_left+vainilla_left == 0)
        System.out.println(sb.toString());
    if(choco_left>0){
        rec(choco_left-1,vainilla_left,sb.append('C'));
        sb.deleteCharAt(sb.length()-1);
    }
    if(vainilla_left>0){
        rec(choco_left,vainilla_left-1,sb.append('V'));
        sb.deleteCharAt(sb.length()-1);
    }
}
```

Generar Helados

Para choco_left=2, vaini_left=1, res=””

→ choco_left=1, vaini_left=1, res=”C”

→ → choco_left=0, vaini_left=1, res=”CC”

→ → → choco_left=0, vaini_left=0, res = “CCV”

→ → choco_left=1, vaini_left=0, res=”CV”

→ → → choco_left=0, vaini_left=0, res=”CVC”

Conclusión

Dejamos a tarea del usuario la manera correcta de imprimir así como la llamada inicial de la función.

Finalmente, esto generará una salida de todos los strings lexicográficamente ordenados si siempre llamamos recursivamente colocando la C antes que la V hasta donde nos sea posible.