Haciendo sushi

Takeshi Kitano está preparando sushi para sus *n* amigos. Sus amigos tienen gustos muy distintos y a cada uno le gusta un tipo de sushi diferente. Cada tipo de sushi usa g_i gramos de pescado. Para que haya más variedad Takeshi quiere comprar *n* tipos de pescado diferentes, uno para cada tipo de sushi. Cada pescado cuesta p_i yenes el gramo.

Dados los g_i y los p_i, ¿cual es el mínimo precio que tiene que pagar Takeshi Kitano para preparar sushi para sus amigos?

Cada tipo de sushi solo puede usar un tipo de pescado y se usan todos los tipos de pescado, uno para cada tipo de sushi.

Estrategia de la solución

Este problema se puede ver como una versión simplificada del problema “Ruleta” del entrenamiento anterior. En este caso tenemos *n* tipos de pescado con precio p_i al gramo y *n* tipos de sushi que necesitan g_i gramos. Tenemos que asignar un tipo diferente de pescado a cada sushi. Al final, el problema se resume en reordenar los p_i y los g_i para minimizar el sumatoria $\sum_{i=1}^{n} p_i * g_i$. Para ello, parecido al problema ruleta tenemos que ordenar los *pi* en orden creciente y los g_i en orden decreciente, o al revés, para conseguir el resultado. Una cosa importante a tener en cuenta es que los valores de p_i y g_i deben ser *long long*, porque su producto no cabe en un *int* normal. En caso de que sean *int* el producto p_i y g_i se guardará temporalmente con *int* y puede perder el valor real.

Solución en Python

```
t = int(input())
for test in range(t):
        n = int(input())
        g = []
        for s in input().split():
                g.append(int(s))
        p = []
        for s in input().split():
                p.append(int(s))
        g = sorted(g)
        p = sorted(p, reverse=True)
        ans = 0
        for i in range(n):
                ans += g[i]*p[i]
        print(ans)
```

Autor de la solución: Izan Beltrán (Miembro del comité organizador)

Solución en C++

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

typedef long long ll;
typedef vector<ll> vll;

int main() {
  int t;
  cin >> t;
  while (t--) {
        int n;
        cin >> n;
        vll G(n), P(n);

        for (int i = 0; i < n; ++i) cin >> G[i];
        for (int i = 0; i < n; ++i) cin >> P[i];

        sort(G.begin(), G.end());
        sort(P.begin(), P.end());

        ll res = 0;
        for (int i = 0; i < n; ++i)
        res += G[i]*P[n-i-1];

        cout << res << endl;
  }
}
```

Autor de la solución: Cesc Folch (Miembro del comité organizador)

Lomo queso

Luís trabaja en un bar preparando bocadillos calientes. Desde hamburguesas hasta bocadillos de tortilla, nada escapa a su legendaria habilidad. Sin embargo, su especialidad es el lomo queso, un bocadillo que consta de dos filetes de lomo acompañados de un par de lonchas de queso. Puesto que el bar es muy transcurrido y poco organizado, *n* personas han pedido un lomo queso a la vez, y Luís quiere preparar todos los bocadillos en el menor tiempo posible, para no tener que soportar los gritos de su jefe.

Luís dispone de una plancha para cocinar los filetes, donde tiene espacio suficiente para cocer hasta cuatro filetes de lomo simultáneamente. Cada cara del filete tarda exactamente 20 segundos en estar lista. Dado que Luís tiene una extensa experiencia en la profesión, puedes asumir que una vez ha cocinado el lomo por ambas caras, finaliza y sirve el bocadillo inmediatamente con el queso frio. Calcula cuánto tiempo tardará Luís en satisfacer su demanda.

Estrategia de la solución

Es fácil ver que si *n* es par, el tiempo necesario es *n*20*, y que es mínimo porque en todo momento se está usando la plancha al completo. En caso de tener *n = 3* vemos que podemos cocinarlos en 60 minutos. Sea S_i la cara superior de los filetes de lomo del bocadillo *i*-ésimo y I_i la cara inferior, podemos organizarlo de la manera siguiente: *(I_1, I_2) (S_1, I_3) (S_2, S_3)*. Y vemos que, efectivamente el menor tiempo para cocinar los 3 bocadillos es 60 minutos. Con esto, si tenemos *n* impar, podemos expresar *n = (n-3) + 3*, donde *(n-3)* es par, y por tanto la solución es *20*(n-3) + 60*. El único caso que falta es el de *n = 1*, que es trivialmente 40.

Solución en Python

```python
import sys

for line in sys.stdin:
	n = int(line)
	if n == 1:
		print(40)
	else:
		print(20*n)
```

Autor de la solución: Izan Beltrán (Miembro del comité organizador)

Solución en C++

```cpp
#include <iostream>

using namespace std;

typedef long long ll;
```

```
int main() {
  ll n;
  while (cin >> n) {
        if (n == 1) cout << 40 << endl;
        else if (n%2) cout << (n-3)*20 + 60 << endl;
        else cout << n*20 << endl;
  }
}
```

Autor de la solución: Cesc Folch (Miembro del comité organizador)

Distracciones

En las anteriores reuniones de la Organización Independiente de Epizootias, también conocida como la Organización Mundial de Sanidad Animal, se reunían todos en Barcelona para discutir sobre los temas más relevantes. Pero cada año ocurrían varios problemas.

Los asistentes en la reunión nunca consiguen concentrarse en la reunión, se distraen por cualquier tontería. En la última reunión por ejemplo se entretuvieron con un papel en la pared con las siguientes líneas.

1
11
21
1211
111221
312211

Estuvieron mucho tiempo intentando descifrar este duro problema hasta que les explicaron la solución. Cada línea simplemente viene generada leyendo en voz alta la línea anterior. Por ejemplo, la segunda línea es "un uno", la tercera es "dos unos" (y no un once) etc.

Para que no vuelva a ocurrir este problema, la organización te pide que hagas un programa que diga el i-ésimo número de la secuencia, $a_0 = 1$.

Estrategia de la solución

Este problema consiste en implementar lo que se nos pide, es decir, a partir del resultado anterior calcular el siguiente número. Una observación que puede ser útil es que nunca aparecen cuatro o más números iguales consecutivos, entonces los números consisten en palabras formadas por los caracteres *1*, *2* y *3*. Esto es fácil de ver, porque si tenemos la secuencia *1111* por ejemplo, esto significa que teníamos dos grupos consecutivos de *1*s, pero tal y como se describe el algoritmo deberían ser un único grupo.

Solución en Python

```
n = int(input())
x = [1]
for i in range(n):
	last = x[0]
	cnt = 1
	y = []
	for j in range(1,len(x)):
		if x[j] == last:
			cnt += 1
		else:
			y.append(cnt)
			y.append(last)
			last = x[j]
```

```
                cnt = 1
        y.append(cnt)
        y.append(last)
        x = y
for v in x:
        print(v, end='')
print() # line break
```

Autor de la solución: Izan Beltrán (Miembro del comité organizador)

Solución en C++

```
#include <iostream>
#include <vector>

using namespace std;

typedef vector<string> vs;

int main() {
  int n;
  cin >> n;
  string b = "1";
  for (int i = 1; i <= n; ++i) {
        string a = "";
        int cont = 0;
        char val = '0';
        for (int j = 0; j < (int)b.size(); ++j) {
                if (b[j] == val) ++cont;
                else {
                        if (cont) {
                                a.push_back(cont + '0');
                                a.push_back(val);
                        }
                        cont = 1;
                        val = b[j];
                }
        }
        a.push_back(cont + '0');
        a.push_back(val);
        b = a;
  }
  cout << b << endl;
}
```

Autor de la solución: Cesc Folch (Miembro del comité organizador)

Monedas

Baq tiene n monedas, cada una con un valor m_i. A Baq no le gusta que las máquinas donde compra los billetes de metro le devuelvan cambio, por eso quiere saber si con sus n monedas puede pagar con exactitud cualquier cantidad.

¿Cual es la mínima cantidad de dinero que Baq no puede pagar con exactitud con sus n monedas?

Estrategia de la solución

Si ordenamos las monedas en orden creciente, es fácil ver que todos los valores menores a m_i tienen que ser posibles con las monedas m_j con $j < i$, en caso que la respuesta sea mayor que m_i. Otra observación importante es que si se cumple lo anterior y con las $(i-1)$ primeras monedas podemos formar todos los valores del intervalo $[0,k]$ con $k + 1 >= m_i$, entonces con las i primeras monedas podemos formar todos los valores del intervalo $[0,k+m_i]$. En caso de coger 0 monedas el intervalo es $[0,0]$, si empezamos con este intervalo y lo vamos aumentando mientras se cumpla $k + 1 >= m_i$, cuando lleguemos a un caso en que $k + 1 < m_i$ o ya hayamos cogido todas las monedas tendremos que el resultado es $k + 1$.

Solución en Python

```python
t = int(input())
for test in range(t):
	n = int(input())
	m = []
	for s in input().split():
		m.append(int(s))
	m = sorted(m)
	suma = 0
	for i in range(len(m)):
		if suma + 1 < m[i]:
			break
		suma += m[i]
	print(suma + 1)
```

Autor de la solución: Izan Beltrán (Miembro del comité organizador)

Solución en C++

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

typedef long long ll;
typedef vector<int> vi;

int main() {
```

```
  int t;
  cin >> t;
  while (t--) {
        int n;
        cin >> n;

        vi M(n);
        for (int i = 0; i < n; ++i) cin >> M[i];
        sort(M.begin(), M.end());

        ll k = 0;
        for (int i = 0; i < n; ++i) {
               if (k + 1 < M[i]) break;
               K += M[i];
        }

        cout << k+1 << endl;
  }
}
```

Autor de la solución: Cesc Folch (Miembro del comité organizador)

El exámen muy difícil

Después de pasar una hora y media en el examen final de Matemáticas elucubrando en vano las respuestas a las preguntas tan diabólicas que han salido, decides que es hora de rendirte, inventarte las respuestas que quedan, entregar y rezar por el aprobado. Afortunadamente para ti, el examen es de tipo test y cada pregunta tiene tres posibles opciones: A, B o C y algunas de las respuestas ya las conoces. Cada acierto vale un punto y los fallos no restan. Recuerdas de los exámenes de prueba que hiciste ayer que el profesor jamás pone dos preguntas consecutivas con la misma respuesta, es decir, que si la pregunta 1 es la A, entonces la pregunta 2 no puede ser la A. Tu objetivo es rellenar el examen de tal forma que maximices el número de puntos que sacarás en el peor de los casos.

Estrategia de la solución

Las demostraciones de este problema se dejan como ejercicio. Las letras que ya sabemos son
aciertos que podemos asegurar, si no tenemos dos *?* consecutivos podemos ver que si el *?* está a principio o final de palabra no podemos asegurar el acierto. Si está entre dos letras iguales tampoco, pero si está entre dos letras diferentes si. En caso de que pueda haber más de un *?* consecutivo tenemos que si el grupo consecutivo está a principio o a final de palabra no podemos asegurar el acierto, pero que si está entre dos letras y el grupo tiene dos o más *?* consecutivos podemos asegurar solo un acierto. A partir de aquí es fácil programar la solución. Si tenéis dudas sobre cómo enfocar la demostración pedid ayuda en Discord.

Solución en C++

```
#include <iostream>

using namespace std;

int main() {
  int t;
  cin >> t;
  while (t--) {
        int n;
        string a;
        cin >> n >> a;
        int res = 0;
        for (int i = 0; i < n; ++i)
               if (a[i] != '?') ++res;
        for (int i = 1; i + 1 < n; ++i)
               if (a[i] == '?' and a[i-1] != '?' and a[i-1] != a[i+1])
                      ++res;
        if (res and n >= 2 and a[n-1] == a[n-2] and a[n-1] == '?') --res;
```

```
        cout << res << endl;
    }
}
```

Autor de la solución: Cesc Folch (Miembro del comité organizador)