Números FizzBuzz

Dado un entero *n*, escribid todos los enteros de 1 a *n*, pero en caso de que el número sea múltiplo de 3 escribid 'Fizz' en lugar del número, si es múltiplo de 5 escribid 'Buzz' y si es múltiplo de 3 y 5 escribid 'FizzBuzz'.

Estrategia de la solución

En este problema solo se pedía una implementación sencilla, no tenía dificultad algorítmica.

Solución en Python

```python
t = int(input())
for tc in range(t):
    n = int(input())
    for i in range(1,n+1):
        if i%15 == 0:
            print("FizzBuzz")
        elif i%3 == 0:
            print("Fizz")
        elif i%5 == 0:
            print("Buzz")
        else:
            print(i)
```

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

Solución en C++

```cpp
#include <iostream>

using namespace std;

int main() {
  int t;
  cin >> t;
  while (t--) {
        int n;
        cin >> n;
        for (int i = 1; i <= n; ++i) {
        if (i%3 == 0 and i%5 == 0)
        cout << "FizzBuzz" << endl;
        else if (i%3 == 0)
        cout << "Fizz" << endl;
        else if (i%5 == 0)
        cout << "Buzz" << endl;
        else cout << i << endl;
        }
  }
}
```

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

Permutando

Dada una permutación de los números del 1 a n: $p_1,p_2...p_n$. Escribid una permutación : de manera que se cumpla para toda con ..

Estrategia de la solución

La manera más fácil de implementar este problema era crear un vector de n elementos, si el *i*-ésimo elemento era *x*, guardamos *v[x]* = *i*. Al escribir la respuesta solo tenemos que escribir *v[i]* en orden creciente de *i*.

Solución en Python

```python
t = int(input())
for tc in range(t):
    n = int(input())
    p = []
    for s in input().split():
        p.append(int(s))
    a = []
    for i in range(n):
        a.append(-1)
    for i in range(n):
        a[p[i]-1] = i+1
    for i in range(n):
        if i == n-1:
            print(a[i], end='')
        else:
            print(a[i], end=' ')
    print('')
```

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

Solución en C++

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

using namespace std;

typedef vector<int> vi;

int main() {
  int t;
  cin >> t;
  while (t--) {
        int n;
        cin >> n;
        vi V(n);
        for (int i = 0; i < n; ++i) {
        int x;
```

```
        cin >> x;
        V[x-1] = i+1;
        }
        for (int i = 0; i < n; ++i) {
        if (i) cout << ' ';
        cout << V[i];
        }
        cout << endl;
  }
}
```

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

Componentes conexas

Dado un grafo, calculad el número de componentes conexas.

Podéis encontrar información útil en los manuales de grafos, DFS y BFS.

Estrategia de la solución

Se recomienda leer los tutoriales que se enlazaban en el problema. Marcamos todos los vértices como no visitados, los recorremos en orden. Cada vez que encontramos un vértice no visitado sumamos uno a la respuesta y ejecutamos un DFS o BFS desde este vértice para marcar toda su componente conexa como visitado.

Solución en Python

```python
def dfs(g, vis, u):
    vis[u] = True
    for i in range(len(g[u])):
        v = g[u][i]
        if not vis[v]:
            dfs(g, vis, v)

t = int(input())
for tc in range(t):
    words = input().split()
    n = int(words[0])
    m = int(words[1])
    g = [] # Lista de n listas de enteros (lista de adyacencia)
    vis = [] # Lista de booleanos para nodos visitados
    for i in range(n):
        g.append([])
        vis.append(False)
    for i in range(m):
        words = input().split()
        u = int(words[0])
        v = int(words[1])
        g[u].append(v)
        g[v].append(u)
    componentes = 0
    for i in range(n):
        if not vis[i]:
            dfs(g, vis, i)
            componentes += 1
    print(componentes)
```

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

Solución en C++

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

```cpp
using namespace std;

typedef vector<int> vi;
typedef vector<vi> vvi;

vvi G;
vi V;

void dfs(int x) {
  V[x] = 1;
  for (auto y : G[x]) {
       if (!V[y]) dfs(y);
  }
}

int main() {
  int t;
  cin >> t;
  while (t--) {
       int n, m;
       cin >> n >> m;
       G = vvi(n);
       V = vi(n, 0);
       while (m--) {
       int a, b;
       cin >> a >> b;
       G[a].push_back(b);
       G[b].push_back(a);
       }
       int res = 0;
       for (int i = 0; i < n; ++i) {
       if (!V[i]) {
       ++res;
       dfs(i);
       }
       }
       cout << res << endl;
  }
}
```

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

Grupos

Hay *n* personas, cada una pertenece a su propio grupo. Cuando dos personas se conocen fusionan sus dos grupos en uno solo. Dado un entero *n* y *q* se harán *n* acciones de tipo 1 o 2.

Tipo 1: Se dan dos enteros *a* y *b* con $0 \leq a,b < n$. Si las personas *a* y *b* no formaban parte del mismo grupo sus dos grupos se fusionan en uno solo.

Tipo 2: Se dan dos enteros *a* y *b* con $0 \leq a,b < n$. Si las personas *a* y *b* forman parte del mismo grupo hay que escribir 'Si', si no 'No'.

Pueden ser de utilidad los manuales 1 y 2.

Estrategia de la solución

Este problema solo nos pedía implementar la estructura de datos descrita en los enlaces del enunciado.

Solución en Python

```python
def find(parent, a):
    if parent[a] != a:
        parent[a] = find(parent, parent[a])
    return parent[a]

def join(parent, a, b):
    a = find(parent, a)
    b = find(parent, b)
    if a != b:
        parent[b] = a

t = int(input())
for tc in range(t):
    nums = input().split()
    n = int(nums[0])
    q = int(nums[1])
    parent = []
    for i in range(n):
        parent.append(i)
    for i in range(q):
        nums = input().split()
        k = int(nums[0])
        a = int(nums[1])
        b = int(nums[2])
        if k == 1:
            join(parent, a, b)
        else:
            if find(parent, a) == find(parent, b):
                print("Si")
```

```
            else:
                print("No")
```

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

Solución en C++

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

using namespace std;

typedef vector<int> vi;

vi parent, size;

int find(int x) {
  if (parent[x] == -1) return x;
  return parent[x] = find(parent[x]);
}

void merge(int x, int y) {
  int x1 = find(x);
  int y1 = find(y);
  if (x1 == y1) return;
  if (size[x1] == size[y1]) {
        size[x1]++;
        parent[y1] = x1;
  }
  else if (size[x1] > size[y1]) {
        parent[y1] = x1;
  }
  else {
        parent[x1] = y1;
  }
}

//Esta función es más lenta pero
//a efectos prácticos funciona más
//o menos igual que la otra pero
//es más fácil y rápida de programar
void merge_slow(int x, int y) {
  int x1 = find(x);
  int y1 = find(y);
  if (x1 != y1)
        parent[x1] = y1;
}

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

```
        parent = vi(n, -1);
        size = vi(n, 0);
        while (q--) {
        int k,a,b;
        cin >> k >> a >> b;
        if (k == 1) merge(a, b);
        else {
        if (find(a) == find(b))
        cout << "Si" << endl;
        else
        cout << "No" << endl;
        }
        }
  }
}
```

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

Monedas 2

Tienes *n* monedas, cada una con un valor m_i, quieres saber cuántos valores diferentes positivos puedes conseguir sumando los valores de algunas monedas.

Pueden ser útiles los manuales de programación dinámica 1 y 2.

Estrategia de la solución

Si sumamos todos los elementos tenemos la suma *S* <= *10000*. Creamos un vector de medidas *D[N][S]*. *D[i][j] = 1* si podemos sumar *j* con los *i* primeros valores o 0 si no. Podemos calcular *D[i+1][j] = max(D[i][j], D[i][j-V[i+1]])* que equivale a mirar los dos casos siguientes, podemos sumar *j* con los *i+1* primeros elementos si podemos sumar *j* con los *i* primeros elementos o si podemos sumar *j-V[i+1]* con los *i* primeros elementos *(V[i+1]* es el *i+1*-ésimo elementos) y luego sumar *V[i+1]*.

Solución en Python

```python
t = int(input())
for tc in range(t):
    n = int(input())
    m = []
    for s in input().split():
        m.append(int(s))
    max_sum = 0
    for i in range(n):
        max_sum += m[i]
    dp = []
    for i in range(n+1):
        dp.append([])
        for j in range(max_sum+1):
            dp[i].append(False)
    # dp[i][j] -> Es posible llegar exactamente al valor j
    # usando algunas de las i primeras monedas?
    dp[0][0] = True
    for i in range(1,n+1):
        for j in range(max_sum+1):
            if dp[i-1][j]:
                if j+m[i-1] <= max_sum:
                    dp[i][j+m[i-1]] = True # Usar la i-ésima moneda (1-indexed)
                dp[i][j] = True # No usarla
    cnt = 0
    for j in range(1, max_sum+1):
        if dp[n][j]:
            cnt += 1
    print(cnt)
```

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

Solución en C++

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

using namespace std;

typedef vector<int> vi;
typedef vector<vi> vvi;

int main() {
  int t;
  cin >> t;
  while (t--) {
        int n;
        cin >> n;
        vi V(n);
        int tot = 0;
        for (int i = 0; i < n; ++i) {
        cin >> V[i];
        tot += V[i];
        }
        vvi D(n+1, vi(tot+1, 0));
        D[0][0] = 1;
        for (int i = 0; i < n; ++i) {
        for (int j = 0; j < V[i]; ++j) {
        D[i+1][j] = D[i][j];
        }
        for (int j = V[i]; j <= tot; ++j) {
        D[i+1][j] = max(D[i][j], D[i][j-V[i]]);
        }
        }
        int res = 0;
        for (int i = 1; i <= tot; ++i) res += D[n][i];
        cout << res << endl;
  }
}
```

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