Envío 79442 — 19/10 10:53

AC

MARP48 | JIANUO WEN HU
MARP16 | PEDRO LEÓN MIRANDA

Nota: 9

CicloDirigido.h

```
10  using Camino = deque<int>;
11
12  class CicloDirigido
13  {
14          std::vector<bool> visit; // visit[v] = ¿se ha alcanzado a v en el dfs?
15          std::vector<bool> apilado; // apilado[v] = ¿está el vértice v en la pila?
16          Camino _ciclo; // ciclo dirigido (vacío si no existe)
17          bool hayciclo;
18
19          void dfs(Digrafo const& g, int v) { //O(V + A) donde V son los vertices y A las
            ↪  aristas
20                  apilado[v] = true;
21                  visit[v] = true;
22                  for (int w : g.ady(v)) {
23                          if (!visit[w]) { // encontrado un nuevo vértice, seguimos
24                                  dfs(g, w);
25                          }
26                          else if (apilado[w]) { // hemos detectado un ciclo
27                                  // se recupera retrocediendo
28                                  hayciclo = true;
29                          }
30                  }
31                  apilado[v] = false;
32          }
33  public:
34          CicloDirigido(Digrafo const& g) : visit(g.V(), false),
35                  apilado(g.V(), false), hayciclo(false) {
36                  dfs(g, 0);
37          }
38          bool hayCiclo() const { return hayciclo; }
39          Camino const& ciclo() const { return _ciclo; }
40
41          bool visited(int v) const {
42                  return visit[v];
43          }
44
45  };
```

Necronomicon.cpp

```
2   /*
3     *
4     * MARP48 Jianuo Wen
5     * MARP16 Pedro León Mirando
6     *
7   */

17  /*
18   Escribe aquí un comentario general sobre la solución, explicando cómo
19   se resuelve el problema y cuál es el coste de la solución, en función
20   del tamaño del problema.
```

```cpp
/*
 //O(V + A) donde V son los vertices y A las aristas
 Se guardan las instrucciones en un grafo. Depdendiendo de la instrucción se conecta de una manera distinta. El grafo tiene
 L + 1 vértices. Se comprueba si el grafo presenta algún ciclo alcanzable desde el primer vértice y se comprueba si se llega
 al final del grafo.

 Para ello hemos modificado la clase cicloDirigido y hemos hecho que la búsqueda no para cuando se encuentre un ciclo
*/

bool resuelveCaso() {

    int L;
    cin >> L;
    if (!cin)
        return false;

    // leer el resto del caso y resolverlo
    char inst;
    Digrafo g(L + 1);
    for (int i = 0; i < L; ++i) {
        cin >> inst;
        if (inst == 'J') {
            int w;
            cin >> w;
            g.ponArista(i, w - 1);
        }
        else if (inst == 'C') {
            int w;
            cin >> w;
            g.ponArista(i, i + 1);
            g.ponArista(i, w - 1);
        }
        else { //es A
            g.ponArista(i, i + 1);
        }
    }

    CicloDirigido cd(g);

    if (cd.hayCiclo() && cd.visited(L)) cout << "A VECES\n";
    else if (cd.hayCiclo() && !cd.visited(L)) cout << "NUNCA\n";
    else cout << "SIEMPRE\n";

    return true;
}
```

Digrafo.h