Alejandro Garcia Carballo 1423957
Marc Martí Asensio 1424859

Proves de caixa blanca

En el nostre codi, hem utilitzat les proves de caixa blanca per testejar la funció comprovarCombinacio() de la classe Solució, ja que es la única funció del programa que té una certa complexitat i té sentit realitzar aquests tests.

- **Control flow**: s'ha controlat el flow amb el eclemma.
- **Statement coverage**: totes les sentencies de la funció s'executen com a mínim una vegada.

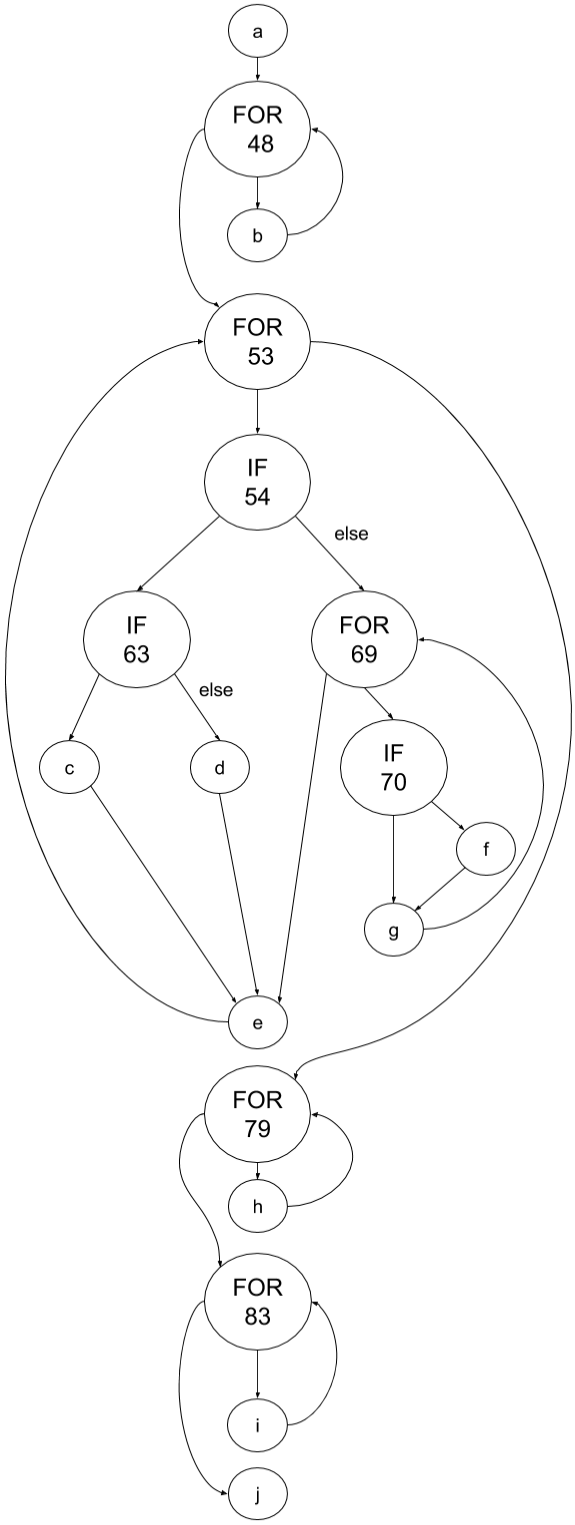
```
40    public int[] comprovarCombinacio(int[] combinacio) {
41        int[] correctes = new int[nNombres];
42        Arrays.fill(correctes, 0);
43        int[] vegadesSurtNombreSolucio = new int[8];    // Conta quantes vegades surt un nombre en la solució
44        Arrays.fill(vegadesSurtNombreSolucio, 0);
45        int contadorNombreIPos = 0;     // contador de numeros iguals amb la mateixa posicio
46        int contadorNombre = 0;     // contador de numeros amb mateix Nombre, posicio diferent
47
48        for (int i = 0; i < nNombres; i++) {        // Conta les vegades que surt cada nombre de la solució
49            vegadesSurtNombreSolucio[this.solucio[i] - 1] += 1;
50        }
51
52        for (int i = 0; i < nNombres; i++) {
53            if (combinacio[i] == this.solucio[i]) {
54                contadorNombreIPos++;
55
56                // Aquesta comprobació controla que no es produeixi el següent error.
57                // Si tenim la solució:     1 2 3 3
58                // I donem la combinació:   3 3 3 5
59                // Com que el codi mira la combinació de esquerra a dreta, si no fessim aquesta comprobació ens retornaria
60                // el següent:  2 1 1 1, ja que quan arriba al tercer 3 el conta encara que ja haguem arribat al nombre de vegades
61                // que apareix el valor 3 en la solució. Amb aquest if-else, obtindrem el següent:    2 1 0 0, que es el resultat correcte.
62
63                if (vegadesSurtNombreSolucio[combinacio[i] - 1] != 0) {
64                    vegadesSurtNombreSolucio[combinacio[i] - 1] -= 1;
65                } else {
66                    contadorNombre -= 1;
67                }
68            } else {
69                for (int j = 0; j < nNombres; j++) {
70                    if (combinacio[i] == this.solucio[j]) {
71
72                        // Aquesta comprobació controla que no es produeixi el següent error.
73                        // Si tenim la solució:     1 2 3 3
74                        // I donem la combinació:   3 3 2 2
75                        // Com que el codi mira la combinació de esquerra a dreta, si no fessim aquesta comprobació ens retornaria
76                        // el següent:  1 1 1 1, ja que quan arriba al segon 2 el conta encara que ja haguem arribat al nombre de vegades
77                        // que apareix el valor 2 en la solució. Amb aquest if, obtindrem el següent:    1 1 1 0, que es el resultat correcte.
78                        if (vegadesSurtNombreSolucio[combinacio[i] - 1] != 0) {
79                            contadorNombre++;
80                            vegadesSurtNombreSolucio[combinacio[i] - 1] -= 1;
81                            j = nNombres;
82                        }
83                    }
84                }
85            }
86        }
87
88        for (int i = 0; i < contadorNombreIPos; i++) {
89            correctes[i] = 2;
90        }
91
92        for (int i = 0; i < contadorNombre; i++) {
93            correctes[i+contadorNombreIPos] = 1;
94        }
95
96        return correctes;
```

Statement coverage de la funció comprovarCombinacio de Solucio.java

Alejandro Garcia Carballo 1423957
Marc Martí Asensio 1424859

- **Decision coverage**: totes les decisions prenen el valor true i false al menys una vegada cadascun. Aquest test es realitza mitjançant tots els test de la funció, ja que amb les diferents combinacions testejades ja es compleixen totes les condicions.
- **Condition coverage**: les conditions de la funció només tenen una decisió, així que pel punt anterior ja es compleix el Condition coverage.
- **Loop testing**: hem agafat els bucles de la funció i els hem fet recórrer per diferents valors:
    - **Bucles nNombres** (for-48, for-53 i for-69): des de 0 fins a 6 (ja que aquest és el nombre màxim de xifres que pot tenir una combinació en el nivell de dificultat Difícil).
    - **Bucle contadorNombreIPos** (for-88): des de 0 fins a 6, igual que per els bucles anteriors.
    - **Bucle contadorNombre** (for-92): des de 0 fins a 6, igual que per els bucles anteriors.
- **Path coverage**: les proves realitzades cobreixen tots els paths independents de la funció.
    1. a, for-48, b, for-48, for-53, if-54, if-63, c, e, for-53, for-88, h, for-88, for-92, j (**test comb_0**).
    2. a, for-48, b, for-48, for-53, if-54, if-63, d, e, for-53, for-88, h, for-88, for-92, i, for-92, j (**test comb_14**).
    3. a, for-48, b, for-48, for-53, if-54, for-69, if-70, g, for-69, e, for-53, for-88, for-92, j (**test comb_2**).
    4. a, for-48, b, for-48, for-53, if-54, for-69, if-70, f, g, for-69, e, for-53, for-88, for-92, i, for-92, j (**test comb_1**)

Alejandro Garcia Carballo 1423957
Marc Martí Asensio 1424859



Graf de fluxe de la funció comprovarCombinacio de Solucio.java