Programa 01

Diego Méndez Medina

- **Alcanzabilidad**

1. Dar su forma canónica

   Al tener una grafica no dirigida $G = (V, A)$, decimos que dos vértices, s y t, son *alcanzables* cuando existe un camino que no repite vértices de s a t en G.

   Enunciado de optimización: Sea $G = (V, A)$ una gráfica simple no dirigida, y s y t dos vértices diferentes. Determinar el camino de s a t con el mínimo número de aristas.

   Enunciado de decisión: Sea $G = (V, A)$ una gráfica simple no dirigida, s y t dos vértices diferentes y k un entero positivo. Nos preguntamos, ¿Existe un camino de s a t de a lo más k vértices?.

2. Diseñar un algoritmo no-determinístico polinomial.

   Asusimos la presencia de una gráfica aleatoria y el algoritmo no-determinista es el siguiente:

   a) Seleccionamos dos vértices s y t.
   b) Inicializamos un conjunto de aristas vacio, M.
   c) Iteramos sobre las aristas, y generamos un numero “aleatorio” r, si es par agregamos la arista a M has juntar las k aristas.

   Para verificar si la solución propuesta se debe hacer lo siguiente:

   a) Lo primero es verificar que s y t figuren en alguna arista, en caso de ser correcto continuamos si no regresamos `NO`.
   b) Para todas las aristas donde figure s hay que seguir el camino que nos lleve dentro del conjunto solucion, en caso de que el camino se “rompa” en algúna arista regresamos `NO`.
   c) Regresamos `Sí`, si encontramos a t antes de completar los k vértices.

3. Implementación

   Para la implementación se utilizó C, se fijo $k = 10$, 15 aristas y 11 vértices.

   Para compilar hacer cualquiera de las dos opciones:

```
src/$ make compile_alcance

src/$ gcc -o alcance alcance.c grafica.c
```

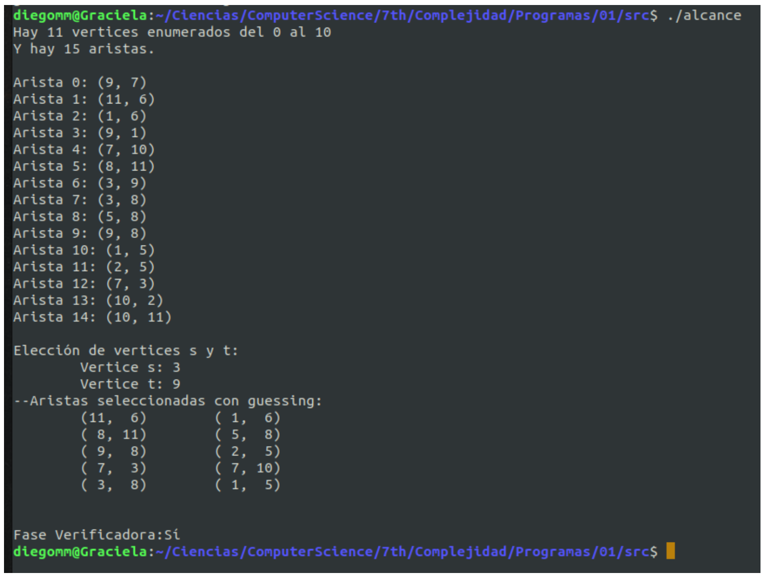
   Para ejecutar:

```
src/$ ./alcance
```

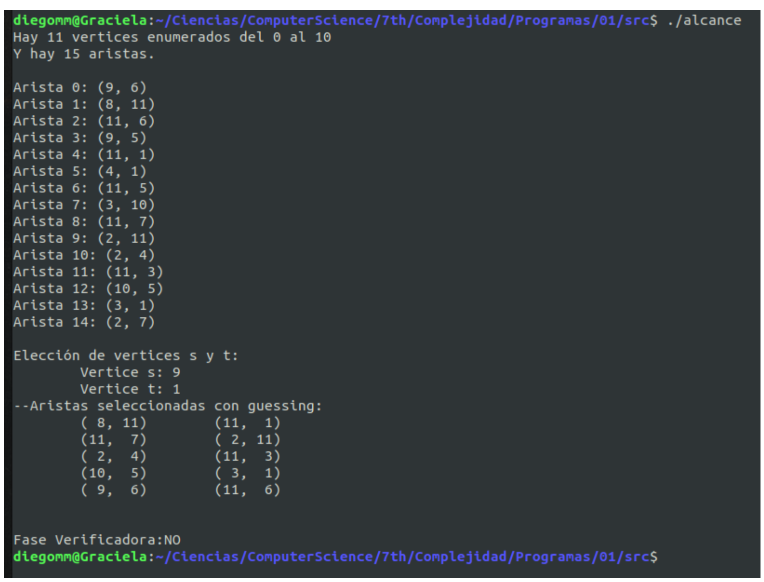
Para compilar **Y** ejecutar:

`src/$ make problema_alcance`

4. Capturas de ejecución:



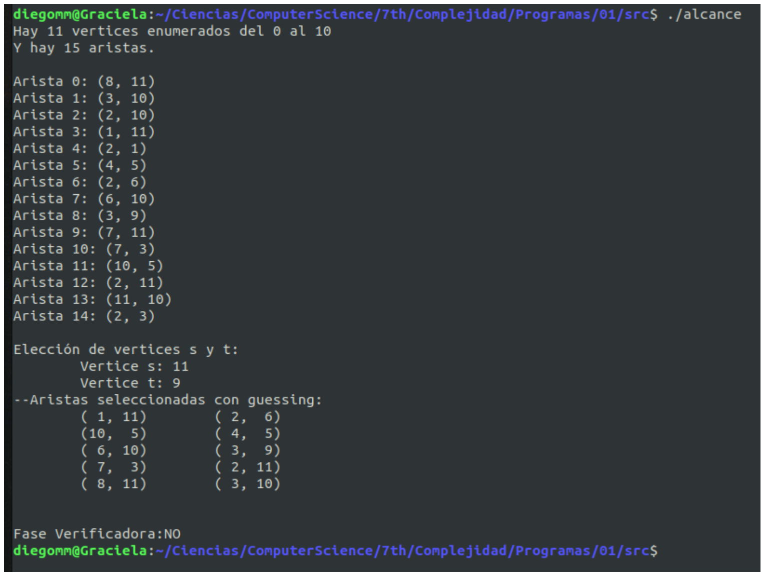
```
diegomm@Graciela:~/Ciencias/ComputerScience/7th/Complejidad/Programas/01/src$ ./alcance
Hay 11 vertices enumerados del 0 al 10
Y hay 15 aristas.

Arista 0: (9, 7)
Arista 1: (11, 6)
Arista 2: (1, 6)
Arista 3: (9, 1)
Arista 4: (7, 10)
Arista 5: (8, 11)
Arista 6: (3, 9)
Arista 7: (3, 8)
Arista 8: (5, 8)
Arista 9: (9, 8)
Arista 10: (1, 5)
Arista 11: (2, 5)
Arista 12: (7, 3)
Arista 13: (10, 2)
Arista 14: (10, 11)

Elección de vertices s y t:
        Vertice s: 3
        Vertice t: 9
--Aristas seleccionadas con guessing:
        (11,  6)        ( 1,  6)
        ( 8, 11)        ( 5,  8)
        ( 9,  8)        ( 2,  5)
        ( 7,  3)        ( 7, 10)
        ( 3,  8)        ( 1,  5)


Fase Verificadora:Sí
diegomm@Graciela:~/Ciencias/ComputerScience/7th/Complejidad/Programas/01/src$
```



```
diegomm@Graciela:~/Ciencias/ComputerScience/7th/Complejidad/Programas/01/src$ ./alcance
Hay 11 vertices enumerados del 0 al 10
Y hay 15 aristas.

Arista 0: (9, 6)
Arista 1: (8, 11)
Arista 2: (11, 6)
Arista 3: (9, 5)
Arista 4: (11, 1)
Arista 5: (4, 1)
Arista 6: (11, 5)
Arista 7: (3, 10)
Arista 8: (11, 7)
Arista 9: (2, 11)
Arista 10: (2, 4)
Arista 11: (11, 3)
Arista 12: (10, 5)
Arista 13: (3, 1)
Arista 14: (2, 7)

Elección de vertices s y t:
        Vertice s: 9
        Vertice t: 1
--Aristas seleccionadas con guessing:
        ( 8, 11)        (11,  1)
        (11,  7)        ( 2, 11)
        ( 2,  4)        (11,  3)
        (10,  5)        ( 3,  1)
        ( 9,  6)        (11,  6)


Fase Verificadora:NO
diegomm@Graciela:~/Ciencias/ComputerScience/7th/Complejidad/Programas/01/src$
```

```
diegomm@Graciela:~/Ciencias/ComputerScience/7th/Complejidad/Programas/01/src$ ./alcance
Hay 11 vertices enumerados del 0 al 10
Y hay 15 aristas.

Arista 0: (6, 3)
Arista 1: (5, 4)
Arista 2: (7, 10)
Arista 3: (4, 1)
Arista 4: (4, 11)
Arista 5: (11, 10)
Arista 6: (3, 8)
Arista 7: (5, 6)
Arista 8: (8, 2)
Arista 9: (8, 4)
Arista 10: (11, 7)
Arista 11: (11, 9)
Arista 12: (6, 1)
Arista 13: (10, 3)
Arista 14: (6, 2)

Elección de vertices s y t:
        Vertice s: 3
        Vertice t: 11
--Aristas seleccionadas con guessing:
        ( 6,  3)        ( 5,  4)
        ( 7, 10)        ( 4,  1)
        ( 5,  6)        ( 8,  4)
        (11,  9)        ( 6,  1)
        ( 6,  2)        ( 3,  8)


Fase Verificadora:Sí
diegomm@Graciela:~/Ciencias/ComputerScience/7th/Complejidad/Programas/01/src$
```

```
diegomm@Graciela:~/Ciencias/ComputerScience/7th/Complejidad/Programas/01/src$ ./alcance
Hay 11 vertices enumerados del 0 al 10
Y hay 15 aristas.

Arista 0: (3, 4)
Arista 1: (4, 9)
Arista 2: (7, 8)
Arista 3: (2, 9)
Arista 4: (7, 3)
Arista 5: (3, 6)
Arista 6: (10, 7)
Arista 7: (2, 3)
Arista 8: (7, 2)
Arista 9: (10, 1)
Arista 10: (10, 3)
Arista 11: (5, 1)
Arista 12: (1, 9)
Arista 13: (8, 10)
Arista 14: (4, 1)

Elección de vertices s y t:
        Vertice s: 6
        Vertice t: 8
--Aristas seleccionadas con guessing:
        ( 3,  4)        ( 4,  9)
        ( 7,  8)        ( 2,  9)
        ( 3,  6)        ( 2,  3)
        ( 7,  2)        (10,  3)
        ( 5,  1)        ( 8, 10)


Fase Verificadora:Sí
diegomm@Graciela:~/Ciencias/ComputerScience/7th/Complejidad/Programas/01/src$
```



- **3-SAT**

  1. Dar su forma canónica

     Al tener una expresión lógica ψ de primer orden en forma CNF, decimos que es *satisfasible* cuando existe una interpretación $I : VAR \to \{true, false\}$, tal que ψ es verdadera.

  2. Diseñar un algoritmo no-determinístico polinomial.

     Suponemos ya existe la ψ en forma CNF con 10 variables conocidas. El algoritmo no deterministico es el siguiente:

     a) Creamos un conjunto de variables que funcionara como modelo.

     b) Iteramos sobre las diez variables, sacamos un número aleatorio, si es par a esa variable se le asigna el valor verdadero.

     Así ya tenemos el ejemplar propuesto, en tiempo lineal checamos la formula *psi* si alguna clausula falla regresamos `NO`, si checamos todas las clausulas entonces regresamos `Sí`.

  3. Implementación

     Para compilar hacer cualquiera de las dos opciones:

     ```
     src/$ make compile_sat
     ```

     ```
     src/$ gcc -o sat 3sat.c FOL.c
     ```

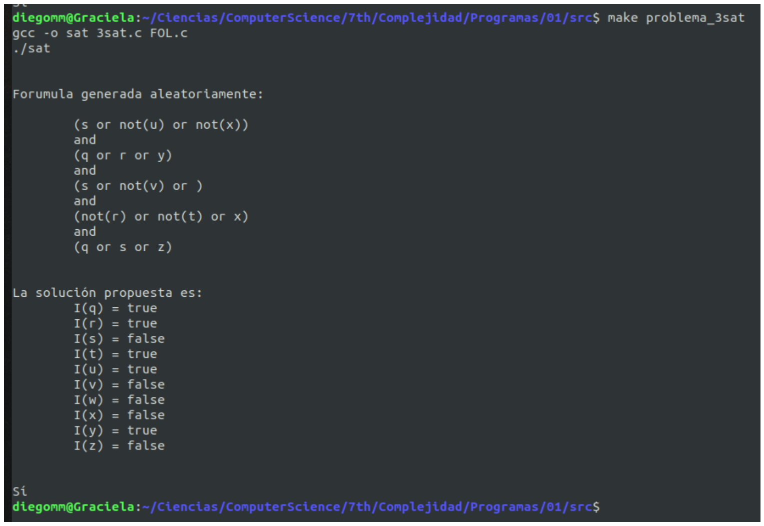
     Para ejecutar:

     ```
     src/$ ./sat
     ```

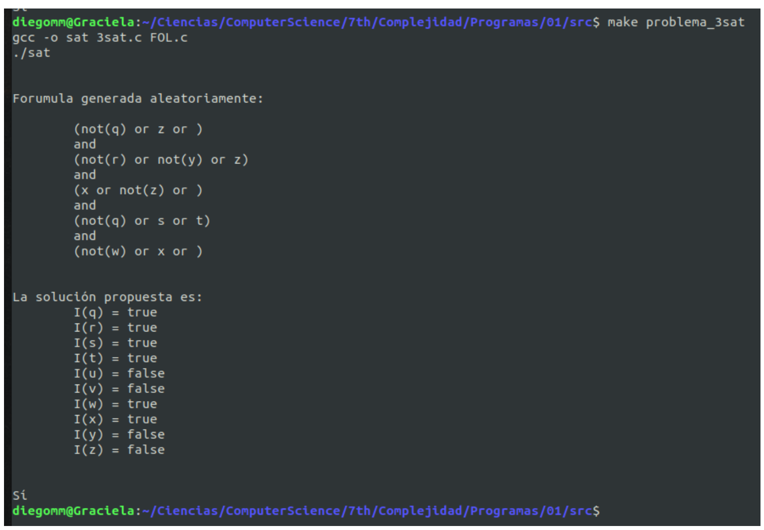
     Para compilar **Y** ejecutar:

```
src/$ make problema_3sat
```

4. Capturas de ejecución:



```
diegomm@Graciela:~/Ciencias/ComputerScience/7th/Complejidad/Programas/01/src$ make problema_3sat
gcc -o sat 3sat.c FOL.c
./sat


Forumula generada aleatoriamente:

        (s or not(u) or not(x))
        and
        (q or r or y)
        and
        (s or not(v) or )
        and
        (not(r) or not(t) or x)
        and
        (q or s or z)


La solución propuesta es:
        I(q) = true
        I(r) = true
        I(s) = false
        I(t) = true
        I(u) = true
        I(v) = false
        I(w) = false
        I(x) = false
        I(y) = true
        I(z) = false


Sí
diegomm@Graciela:~/Ciencias/ComputerScience/7th/Complejidad/Programas/01/src$
```



```
diegomm@Graciela:~/Ciencias/ComputerScience/7th/Complejidad/Programas/01/src$ make problema_3sat
gcc -o sat 3sat.c FOL.c
./sat


Forumula generada aleatoriamente:

        (not(q) or z or )
        and
        (not(r) or not(y) or z)
        and
        (x or not(z) or )
        and
        (not(q) or s or t)
        and
        (not(w) or x or )


La solución propuesta es:
        I(q) = true
        I(r) = true
        I(s) = true
        I(t) = true
        I(u) = false
        I(v) = false
        I(w) = true
        I(x) = true
        I(y) = false
        I(z) = false


Sí
diegomm@Graciela:~/Ciencias/ComputerScience/7th/Complejidad/Programas/01/src$
```

```
diegomm@Graciela:~/Ciencias/ComputerScience/7th/Complejidad/Programas/01/src$ make problema_3sat
gcc -o sat 3sat.c FOL.c
./sat


Forumula generada aleatoriamente:

        (not(w) or x or )
        and
        (not(v) or y or not(z))
        and
        (not(q) or not(w) or )
        and
        (not(q) or u or not(y))
        and
        (q or s or not(y))


La solución propuesta es:
        I(q) = false
        I(r) = true
        I(s) = true
        I(t) = true
        I(u) = false
        I(v) = false
        I(w) = false
        I(x) = true
        I(y) = true
        I(z) = false


Sí
diegomm@Graciela:~/Ciencias/ComputerScience/7th/Complejidad/Programas/01/src$
```

```
diegomm@Graciela:~/Ciencias/ComputerScience/7th/Complejidad/Programas/01/src$ make problema_3sat
gcc -o sat 3sat.c FOL.c
./sat


Forumula generada aleatoriamente:

        (not(r) or t or v)
        and
        (s or not(u) or y)
        and
        (r or not(t) or z)
        and
        (not(q) or not(v) or z)
        and
        (u or not(v) or y)


La solución propuesta es:
        I(q) = false
        I(r) = false
        I(s) = false
        I(t) = false
        I(u) = false
        I(v) = true
        I(w) = true
        I(x) = false
        I(y) = true
        I(z) = false


Sí
diegomm@Graciela:~/Ciencias/ComputerScience/7th/Complejidad/Programas/01/src$
```

```
diegomm@Graciela:~/Ciencias/ComputerScience/7th/Complejidad/Programas/01/src$ make problema_3sat
gcc -o sat 3sat.c FOL.c
./sat


Forumula generada aleatoriamente:

        (not(s) or not(u) or v)
        and
        (q or not(t) or )
        and
        (not(r) or not(t) or w)
        and
        (r or not(x) or not(y))
        and
        (v or not(y) or not(z))


La solución propuesta es:
        I(q) = false
        I(r) = false
        I(s) = true
        I(t) = true
        I(u) = true
        I(v) = true
        I(w) = true
        I(x) = false
        I(y) = true
        I(z) = false


Sí
diegomm@Graciela:~/Ciencias/ComputerScience/7th/Complejidad/Programas/01/src$
```