UNAM

Facultad de Ciencias

Complejidad Computacional
2022-2

Práctica 01

Arroyo Lozano Santiago

October 19, 2022

1 Alcanzabilidad

1.1 Forma Canónica

Una gráfica no dirigida $G = (V, E)$, y dos vértices distinguidos s y t. V es un conjunto de vértices y E es un conjunto de tuplas, que denotan los aristas entre dos vértices.

¿Existe un camino que no repite vértices entre s y t?

1.2 Algoritmo no determinista

Algorithm 1 Alcanzabilidad no determinista

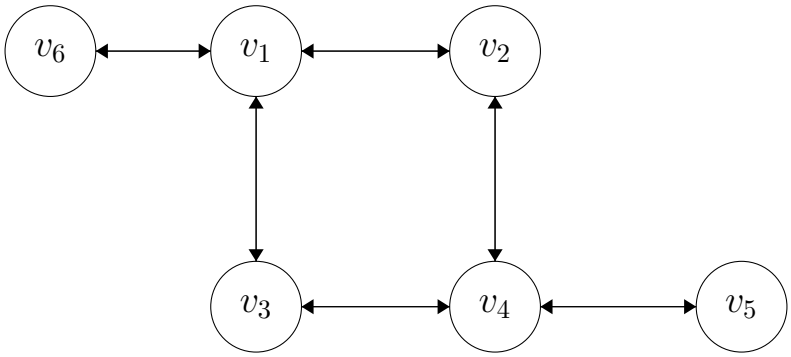
```
1:  procedure ALCANZABILIDAD(G(V, E), s, t)
2:      x ← s
3:      vertices ← V
4:      n ← |V|
5:      for {1, 2, ..., n}
6:          y ← adivina({v_1, v_2, ..., v_n})
7:          if not (x, y) ∈ E
8:              "no"
9:          if  y = t
10:             "sí"
11:         x ← t
```

Lo que hacemos es renombrar nuestro vértice inicial s por x Después elegiremos un vértice al azar y y si existe el arista (x, y) renombraremos a x como y. Así de forma aleatoria iremos buscando un camino desde s a t. Nos detenemos si encontramos a t.

El algoritmo no determinista es polinomial pues a lo más recorremos n vértices. Si la fase adivinadora elige vértices que conformen el camino entonces será una corrida exitosa. Cabe mencionar que la fase adivinadora se repite varias veces pues estamos escogiendo vértices de forma azarosa regularmente. Además, verificamos en cada elección si el camino dado es válido.

1.3 Ejecuciones

Todas las siguientes ejecuciones se hicieron buscando un camino entre v_1 y v_4 con la misma gráfica:




```
El camino recorrido fue
1
2
3
False

[Done] exited with code=0 in 0.133 seconds
```

```
El camino recorrido fue
1
3
6
False

[Done] exited with code=0 in 0.111 seconds
```

```
El camino correcto fue
1
3
True

[Done] exited with code=0 in 0.112 seconds
```

```
El camino recorrido fue
1
2
False

[Done] exited with code=0 in 0.111 seconds
```

```
El camino recorrido fue
1
6
False

[Done] exited with code=0 in 0.117 seconds
```

2 3.SAT

2.1 Forma Canónica

La forma canónica de 3SAT es una expresión en su FNC con variables $x_1, \ldots, x_n$ y cláusulas $C_1, \ldots, C_k$ tal que

$$C_1 \wedge C_2 \wedge \cdots \wedge C_k$$

con $C = \{x_1 \vee x_2 \vee x_3\}$ donde x_i son variables booleanas (exactamente 3 siempre)

2.2 Algoritmo no determinista

Algorithm 2 3SAT no determinista

```
1:  procedure 3SAT(input)
2:      for x_i ∈ {x_1, x_2, ..., x_n}
3:          x_i ← adivina({True, False})  //fase adivinadora. Asignamos valores
    aleatorios
4:      validez ← True
5:      for C ∈ {C_1, C_2, ..., C_k}  //fase verificadora
6:          validezC ← False
7:          for x_i ∈ {x_1, x_2, ..., x_n}
8:              validezC ← validezC ∨ input(x_i) //revisamos si x_i está negada
9:          validez ← validez ∧ validezC
10:     return validez
```

Lo que hacemos es asignar valores aleatorios a las variables para después revisar si cumplen el 3SAT dado en el input. *validez* representa el valor de todo el 3SAT y *validezC* representa el valor de la cláusula que se esté revisando en ese momento, se asume que es falso pues al hacer $\vee$ cambiaría su valor a verdadero. Debe haber al menos una variable verdadera en la cláusula. Por supuesto debemos revisar en *input* si la variable ha sido negada o no.

El algoritmo no determinista es polinomial pues siempre vamos a revisar que se cumplan las k clausulas recorriéndolas linealmente. Por cada cláusula haremos 3 revisiones, una par cada variable en la cláusula. Entonces la complejidad del algoritmo está dada por $O(3k)$.

2.3 Ejecuciones

Todas las siguientes ejecuciones se hicieron una solución válida al siguiente 3SAT

$$(x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3)$$

```
[Running] python -u "c:\Users\santi\Documents\Coding\Python\
Complejidad\src\non_det_algos.py"
La formula es x1 OR x2 OR x3 AND NOT x1 OR NOT x2 OR x3
Se propuso x1=0, x2=0, x3=0
Lamentablemente no es solucion :(

[Done] exited with code=0 in 0.121 seconds
```

```
[Running] python -u "c:\Users\santi\Documents\Coding\Python\
Complejidad\src\non_det_algos.py"
La formula es x1 OR x2 OR x3 AND NOT x1 OR NOT x2 OR x3
Se propuso x3=0, x1=0, x2=1
Es una solucion aceptada!

[Done] exited with code=0 in 0.112 seconds
```

```
[Running] python -u "c:\Users\santi\Documents\Coding\Python\
Complejidad\src\non_det_algos.py"
La formula es x1 OR x2 OR x3 AND NOT x1 OR NOT x2 OR x3
Se propuso x2=1, x1=1, x3=0
Lamentablemente no es solucion :(

[Done] exited with code=0 in 0.12 seconds
```

```
[Running] python -u "c:\Users\santi\Documents\Coding\Python\
Complejidad\src\non_det_algos.py"
La formula es x1 OR x2 OR x3 AND NOT x1 OR NOT x2 OR x3
Se propuso x1=1, x3=0, x2=1
Lamentablemente no es solucion :(

[Done] exited with code=0 in 0.111 seconds
```

```
[Running] python -u "c:\Users\santi\Documents\Coding\Python\
Complejidad\src\non_det_algos.py"
La formula es x1 OR x2 OR x3 AND NOT x1 OR NOT x2 OR x3
Se propuso x2=0, x1=1, x3=0
Es una solucion aceptada!

[Done] exited with code=0 in 0.113 seconds
```

3 Implementación

La práctica se implementó en `python 3`.

El archivo que se debe ejecutar es **practica1.py**. Sin embargo los archivos `obj.py` y `non_det_algos.py` son necesarios para el correcto funcionamiento del programa ya que en ellas se definen ciertas dependencias que usaremos para resolver el problema.

3.1 obj.py

Aquí se encuentran objetos definidos ara ayudarnos a resolver la práctica. Van desde Estructuras de Datos hasta verificadores.

3.2 non_det_algos.py

En este archivo se encuentra la implementación de los algoritmos no deterministas que se plantamos en este documento. Cabe mencionar que para 3SAT, la fase verificadora del algoritmo se desarrolla dentro de un objeto definido en `obj.py`

3.3 practica1.py

En este archivo es donde creamos de forma aleatoria ejemplares para los problemas de alcanzabilidad y 3SAt. Además, corremos los algoritmos no deterministas de ambos para ver si se propuso una respuesta que los resuelva.