Complejidad Computacional

Programa01

Autor: Fausto David Hernández Jasso

September 15, 2022

1 Alcanzabilidad

1.1 Forma Canónica

1.1.1 Ejemplar Genérico

Una gráfica simple no dirigida $G = (V, E)$ donde

- V es el conjunto de vértices.
- E es el conjunto de aristas.

1.1.2 Pregunta de decisión

Dados dos vértices $s, t \in V$. ¿Existe un camino que no repite vértices de s a t en G?

1.2 Algoritmo no-determinístico polinomial

- Determinamos el número de vértices que tendrá la gráfica de manera no determinista. Este número siempre estará entre 10 y 20. Éste número de vértices será n.
- Creamos una matriz M de $n \times n$ para representar a la gráfica G. Inicialmente todas las entradas de la matriz serán inicializadas en 0.
- Creamos aristas de manera no determinista, notemos que sí agregamos la arista (v, u) también debe de estar la arista (u, v). Ésto es que sí la entrada $M[v][u] = 1$ entonces $M[u][v] = 1$.
- Elegimos dos vértices de manera no determinista que serán s y t.
- Construimos un camino de longitud aleatoria donde el ínicio es el vértice s y el fin es el vértice t y los vértices intermedios son elegidos de manera aleatoria.
- Sea $C = \{v_0 = s, v_1, \ldots, v_{k-1}, v_k = t\}$ el camino generado en el paso anterior, entonces comprobamos que v_i es vecino de v_{i+1} para $0 \leq i \leq k - 1$. Sí es cierto lo anterior entonces existe un camino de s a t en G, en otro caso, el candidato que construimos no es un camino de s a t

1.2.1 Capturas de pantalla

```
lechedevainilla@lechedevainilla-G7-7588:~/Semestre2023-1/Complejidad_Computacional/Programas/Programa01$ python3 programa01.py
Vértice 1
Vecinos: [2, 3, 4, 5, 7, 11, 12]
Vértice 2
Vecinos: [1, 4, 6, 8, 9, 10, 11, 12, 16]
Vértice 3
Vecinos: [1, 4, 9, 10, 11, 12, 14, 16]
Vértice 4
Vecinos: [1, 2, 3, 5, 7, 11, 12, 14]
Vértice 5
Vecinos: [1, 4, 7, 10, 12, 14, 15, 16]
Vértice 6
Vecinos: [2, 8, 9, 10, 11, 13, 14]
Vértice 7
Vecinos: [1, 4, 5, 8, 10, 11, 12, 13, 16]
Vértice 8
Vecinos: [2, 6, 7, 9, 10, 12, 13, 14, 15, 16]
Vértice 9
Vecinos: [2, 3, 6, 8, 11, 12, 13, 15, 16]
Vértice 10
Vecinos: [2, 3, 5, 6, 7, 8, 14, 16]
Vértice 11
Vecinos: [1, 2, 3, 4, 6, 7, 9, 12, 13, 15, 16]
Vértice 12
Vecinos: [1, 2, 3, 4, 5, 7, 8, 9, 11, 13, 14, 16]
Vértice 13
Vecinos: [6, 7, 8, 9, 11, 12, 16]
Vértice 14
Vecinos: [3, 4, 5, 6, 8, 10, 12, 15]
Vértice 15
Vecinos: [5, 8, 9, 11, 14, 16]
Vértice 16
Vecinos: [2, 3, 5, 7, 8, 9, 10, 11, 12, 13, 15]
Determinaremos si existe un camino del vértice 12 al vértice 4.
Generando lista de vértices...
Camino candidato: [12, 7, 10, 2, 8, 11, 16, 14, 4, 3, 5]
El candidato [12, 7, 10, 2, 8, 11, 16, 14, 4, 3, 5] no es solución al problema
```

```
lechedevainilla@lechedevainilla-G7-7588:~/Semestre2023-1/Complejidad_Computacional/Programas/Programa01$ python3 programa01.py
Vértice 1
Vecinos: [3, 4, 5, 7, 8, 10, 11, 16, 17]
Vértice 2
Vecinos: [3, 6, 8, 10, 11, 13, 14, 16]
Vértice 3
Vecinos: [1, 2, 5, 6, 8, 10, 11, 12, 13, 14, 15, 16]
Vértice 4
Vecinos: [1, 5, 7, 8, 9, 12, 13, 14, 15]
Vértice 5
Vecinos: [1, 3, 4, 6, 7, 8, 9, 10, 12, 16]
Vértice 6
Vecinos: [2, 3, 5, 7, 8, 9, 11, 12, 13, 14, 15, 16, 17]
Vértice 7
Vecinos: [1, 4, 5, 6, 8, 9, 12, 13, 16, 17]
Vértice 8
Vecinos: [1, 2, 3, 4, 5, 6, 7, 10, 12, 15, 16]
Vértice 9
Vecinos: [4, 5, 6, 7, 10, 12, 16, 17]
Vértice 10
Vecinos: [1, 2, 3, 5, 8, 9, 11, 13, 14, 16, 17]
Vértice 11
Vecinos: [1, 2, 3, 6, 10, 13, 14, 17]
Vértice 12
Vecinos: [3, 4, 5, 6, 7, 8, 9, 13, 15, 16, 17]
Vértice 13
Vecinos: [2, 3, 4, 6, 7, 10, 11, 12, 15, 16, 17]
Vértice 14
Vecinos: [2, 3, 4, 6, 10, 11, 15, 16, 17]
Vértice 15
Vecinos: [3, 4, 6, 8, 12, 13, 14, 16]
Vértice 16
Vecinos: [1, 2, 3, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15]
Vértice 17
Vecinos: [1, 6, 7, 9, 10, 11, 12, 13, 14]
Determinaremos sí existe un camino del vértice 9 al vértice 7.
Generando lista de vértices...
Camino candidato: [9, 3, 5, 4, 7]
El candidato [9, 3, 5, 4, 7] no es solución al problema
```

```
lechedevainilla@lechedevainilla-G7-7588:~/Semestre2023-1/Complejidad_Computacional/Programas/Programa01$ python3 programa01.py
Vértice 1
Vecinos: [2, 3, 6, 7, 8, 10]
Vértice 2
Vecinos: [1, 3, 4, 6, 9, 10]
Vértice 3
Vecinos: [1, 2, 4, 5, 6, 7, 8, 9, 10]
Vértice 4
Vecinos: [2, 3, 5, 7, 8, 9, 10]
Vértice 5
Vecinos: [3, 4, 6, 7, 8, 9, 10]
Vértice 6
Vecinos: [1, 2, 3, 5, 7, 8, 9, 10]
Vértice 7
Vecinos: [1, 3, 4, 5, 6, 8, 10]
Vértice 8
Vecinos: [1, 3, 4, 5, 6, 7, 9]
Vértice 9
Vecinos: [2, 3, 4, 5, 6, 8, 10]
Vértice 10
Vecinos: [1, 2, 3, 4, 5, 6, 7, 9]
Determinaremos sí existe un camino del vértice 10 al vértice 2.
Generando lista de vértices...
Camino candidato: [10, 6, 3, 1, 7]
El candidato [10, 6, 3, 1, 7] sí es solución al problema
```

```
lechedevainilla@lechedevainilla-G7-7588:~/Semestre2023-1/Complejidad_Computacional/Programas/Programa01$ python3 programa01.py
Vértice 1
Vecinos: [2, 4, 5, 6, 8, 9, 10, 12, 13]
Vértice 2
Vecinos: [1, 3, 6, 7, 8, 9, 10, 13]
Vértice 3
Vecinos: [2, 7, 8, 9, 10, 12, 13]
Vértice 4
Vecinos: [1, 5, 7, 9, 10, 12, 13]
Vértice 5
Vecinos: [1, 4, 6, 7, 8, 9, 10, 11]
Vértice 6
Vecinos: [1, 2, 5, 8, 9, 10, 11, 12]
Vértice 7
Vecinos: [2, 3, 4, 5, 8, 9, 10, 12]
Vértice 8
Vecinos: [1, 2, 3, 5, 6, 7, 10, 11, 12]
Vértice 9
Vecinos: [1, 2, 3, 4, 5, 6, 7, 11, 13]
Vértice 10
Vecinos: [1, 2, 3, 4, 5, 6, 7, 8, 13]
Vértice 11
Vecinos: [5, 6, 8, 9, 13]
Vértice 12
Vecinos: [1, 3, 4, 6, 7, 8]
Vértice 13
Vecinos: [1, 2, 3, 4, 9, 10, 11]
Determinaremos sí existe un camino del vértice 11 al vértice 10.
Generando lista de vértices...
Camino candidato: [11, 9, 6, 12, 8, 13, 3, 1]
El candidato [11, 9, 6, 12, 8, 13, 3, 1] no es solución al problema
```

```
lechedevainilla@lechedevainilla-G7-7588:~/Semestre2023-1/Complejidad_Computacional/Programas/Programa01$ python3 programa01.py
Vértice 1
Vecinos: [2, 3, 7, 9, 10]
Vértice 2
Vecinos: [1, 3, 6, 8, 10]
Vértice 3
Vecinos: [1, 2, 4, 9, 10]
Vértice 4
Vecinos: [3, 5, 7, 9]
Vértice 5
Vecinos: [4, 6, 7, 8, 9, 10]
Vértice 6
Vecinos: [2, 5, 7, 8]
Vértice 7
Vecinos: [1, 4, 5, 6, 8, 9, 10]
Vértice 8
Vecinos: [2, 5, 6, 7, 10]
Vértice 9
Vecinos: [1, 3, 4, 5, 7, 10]
Vértice 10
Vecinos: [1, 2, 3, 5, 7, 8, 9]
Determinaremos sí existe un camino del vértice 3 al vértice 5.
Generando lista de vértices...
Camino candidato: [3, 10, 5, 6, 8, 9]
El candidato [3, 10, 5, 6, 8, 9] no es solución al problema
```

2 3-SAT

2.1 Forma Canónica

2.1.1 Ejemplar Genérico

Un conjunto $V = \{v_1, v_2, \ldots, v_n\}$ de n variables y conjunto $C = \{c_1, c_2, \ldots, c_m\}$ de m clausulas sobre V, cada cláusula con a lo más 3 variables.

2.1.2 Pregunta de decisión

¿Existe alguna asignación de valores para las variables de V que hace que las clásulas de C sean todas verdaderas y por ende la expresión sea verdadera?

2.2 Algoritmo no-determinístico polinomial

- Construimos de manera aleatoria 5 cláusulas. Con 3 literales exactamente cada una.
- Elegimos de manera aleatoria su valor, ya sea $True$ o $False$
- Sí la literal x tiene el valor de v entonces $\not x$ tendrá el valor $\not v$ y viceversa.
- Evaluamos la expresión.
- Sí se satisfació la expresión devolvemos que la expresión bajo los estados asignados es verdadera, en otro caso la expresión es falsa.

2.2.1 Capturas de pantalla

```
lechedevainilla@lechedevainilla-G7-7588:~/Semestre2023-1/Complejidad_Computacional/Programas/Programa01$ python3 programa01.py
Evaluando la expresión con la asignación de estados
Expresión
( ~a || z || z ) & ( y || y || ~u ) & ( ~y || ~x || ~y ) & ( ~x || ~a || ~z ) & ( ~c || a || y )
Asignación de estados de las variables:
Variable: ~a Estado: False
Variable: z Estado: False
Variable: z Estado: False
Variable: y Estado: False
Variable: y Estado: False
Variable: ~u Estado: False
Variable: ~y Estado: True
Variable: ~x Estado: True
Variable: ~y Estado: True
Variable: ~x Estado: True
Variable: ~a Estado: False
Variable: ~z Estado: True
Variable: ~c Estado: False
Variable: a Estado: True
Variable: y Estado: False
La expresión no se satisface dado el conjunto de estados de las variables.
```

```
lechedevainilla@lechedevainilla-G7-7588:~/Semestre2023-1/Complejidad_Computacional/Programas/Programa01$ python3 programa01.py
Evaluando la expresión con la asignación de estados
Expresión
( v || v || ~x ) & ( z || u || z ) & ( b || u || ~z ) & ( b || w || c ) & ( ~d || ~u || v )
Asignación de estados de las variables:
Variable: v Estado: True
Variable: v Estado: True
Variable: ~x Estado: False
Variable: z Estado: True
Variable: u Estado: False
Variable: z Estado: True
Variable: b Estado: False
Variable: u Estado: False
Variable: ~z Estado: False
Variable: b Estado: False
Variable: w Estado: True
Variable: c Estado: True
Variable: ~d Estado: True
Variable: ~u Estado: True
Variable: v Estado: True
La expresión no se satisface dado el conjunto de estados de las variables.
```

```
lechedevainilla@lechedevainilla-G7-7588:~/Semestre2023-1/Complejidad_Computacional/Programas/Programa01$ python3 programa01.py
Evaluando la expresión con la asignación de estados
Expresión
( c || ~c || z ) & ( ~z || u || w ) & ( ~u || w || ~v ) & ( ~a || ~b || ~v ) & ( y || u || b )
Asignación de estados de las variables:
Variable: c Estado: True
Variable: ~c Estado: False
Variable: z Estado: True
Variable: ~z Estado: False
Variable: u Estado: False
Variable: w Estado: False
Variable: ~u Estado: True
Variable: w Estado: False
Variable: ~v Estado: True
Variable: ~a Estado: True
Variable: ~b Estado: True
Variable: ~v Estado: True
Variable: y Estado: False
Variable: u Estado: False
Variable: b Estado: False
La expresión no se satisface dado el conjunto de estados de las variables.
```

```
lechedevainilla@lechedevainilla-G7-7588:~/Semestre2023-1/Complejidad_Computacional/Programas/Programa01$ python3 programa01.py
Evaluando la expresión con la asignación de estados
Expresión
( u || z || a ) & ( v || ~z || y ) & ( ~b || u || x ) & ( u || x || c ) & ( ~c || ~d || ~w )
Asignación de estados de las variables:
Variable: u Estado: True
Variable: z Estado: False
Variable: a Estado: True
Variable: v Estado: True
Variable: ~z Estado: True
Variable: y Estado: False
Variable: ~b Estado: True
Variable: u Estado: True
Variable: x Estado: True
Variable: u Estado: True
Variable: x Estado: True
Variable: c Estado: False
Variable: ~c Estado: True
Variable: ~d Estado: False
Variable: ~w Estado: False
La expresión sí se satisface dado el conjunto de estados de las variables.
```

```
lechedevainilla@lechedevainilla-G7-7588:~/Semestre2023-1/Complejidad_Computacional/Programas/Programa01$ python3 programa01.py
Evaluando la expresión con la asignación de estados
Expresión
( ~d || d || ~a ) & ( ~b || x || ~u ) & ( u || w || c ) & ( ~z || ~y || ~x ) & ( b || ~x || ~d )
Asignación de estados de las variables:
Variable: ~d Estado: True
Variable: d Estado: False
Variable: ~a Estado: True
Variable: ~b Estado: True
Variable: x Estado: True
Variable: ~u Estado: False
Variable: u Estado: True
Variable: w Estado: True
Variable: c Estado: True
Variable: ~z Estado: True
Variable: ~y Estado: True
Variable: ~x Estado: False
Variable: b Estado: False
Variable: ~x Estado: False
Variable: ~d Estado: True
La expresión sí se satisface dado el conjunto de estados de las variables.
```