x

Universidad Nacional
Autónoma de México

Facultad de Ciencias

Complejidad Computacional

2023-1

Programa 01

David Hernández Uriostegui
Liera Montaño Miguel Ángel

16 de septiembre de 2022

1 Actividades

1 Considera los siguientes problemas:

a) **Alcanzabilidad:** Dada una gráfica no dirigida $G = (V, E)$, con dos vértices distinguidos s y t. ¿Existe un camino que no repite vértices de s a t en G?

a) Dar su forma canónica.

◇ **Ejemplar Genérico** Una gráfica simple no dirigida $G = (V, E)$

Enunciado de Optimización Determinar el camino de s a t con la cantidad mínima de vértices.

Enunciado genérico Una gráfica simple no dirigida $G = (V, E)$ con un entero positivo k.

Pregunta de decisión ¿Existe un camino (sucesión de vértices) de G de s a t de cardinalidad a lo más k?

b) Diseñar un algoritmo no-determinístico polinomial.

Es lógico pensar que la solución más rápida para este problema es crear un algoritmo que genere un camino al azar (entre comillas) empezando con s, visitando cada uno de los vecinos de dicho vértice, escogiendo uno para agregarlo al camino propuesto y repetir hasta que el nodo actual sea t.

Nuestra condición primitiva *nd* aquí será verificar que el conjunto de nodos visitados hasta el momeneto no sea mayor o igual a la lista de vecino del nodo tomado al azar.

```
1
2  def reachability(s, t, graph):
3      current_node = graph.vertices[s.id]
4
5      path = [current_node]
6      seen = set([current_node.id])
7
8      while graph.vertices[current_node.id].neighbours and
       current_node != t:
9          while current_node in path:
10             seen.add(current_node.id)
11             current_node = random.choice(graph.vertices[
       current_node.id].neighbours)
12
13             if len(seen) >= len(graph.vertices[current_node.id].
       neighbours):  # Usamos esto como primitva nd
14                 break
15
16         if current_node not in path:
17             path.append(current_node)
18             seen.clear()
19
```

```
20          if len(seen) >= len(graph.vertices[current_node.id].
    neighbours): # Usamos esto como primitva nd
21              break
22
23      return t == path[-1], [v.id for v in path]
```

Listing 1: Algoritmo para Alcanzabilidad

c) Implementar el algoritmo diseñado.

```
davidhernandez@Davids-MacBook-Pro Programa01 % python3 Alcanzabilidad.py
Gráfica generada con 12 vértices generada: aleatoriamente:

0 : [1, 2, 3, 6, 8, 9, 11]
1 : [0, 4, 6]
2 : [0, 4, 5, 6, 10]
3 : [0, 4, 5, 6, 9, 10]
4 : [1, 2, 3, 6, 7, 8, 10, 11]
5 : [2, 3, 8, 9, 11]
6 : [0, 1, 2, 3, 4, 8, 9]
7 : [4, 10, 11]
8 : [0, 4, 5, 6, 10, 11]
9 : [0, 3, 5, 6, 10, 11]
10 : [2, 3, 4, 7, 8, 9, 11]
11 : [0, 4, 5, 7, 8, 9, 10]
--------------------------------------
ALCANZABILIDAD
Nodo s = 8
Nodo t = 6

El candidato propuesto es:
[8, 11, 4, 1, 10, 6]


El candidato propuesto sí es solución
```

```
davidhernandez@Davids-MacBook-Pro Programa01 % python3 Alcanzabilidad.py
Gráfica generada con 20 vértices generada: aleatoriamente:

0 : [4, 5, 8, 11, 12, 14, 17, 18]
1 : [2, 3, 5, 6, 7, 8, 11, 13, 15, 18, 19]
2 : [1, 3, 4, 5, 6, 7, 9, 12, 13, 16]
3 : [1, 2, 8, 11, 12, 15, 16, 17, 18, 19]
4 : [0, 2, 5, 7, 8, 13, 14, 16, 17, 18, 19]
5 : [0, 1, 2, 4, 6, 7, 8, 12, 13, 14, 17, 18, 19]
6 : [1, 2, 5, 7, 10, 14, 15, 18, 19]
7 : [1, 2, 4, 5, 6, 8, 9, 12, 14, 16, 18, 19]
8 : [0, 1, 3, 4, 5, 7, 11]
9 : [2, 7, 10, 15, 17, 18, 19]
10 : [6, 9, 13, 15, 17]
11 : [0, 1, 3, 8, 12, 14, 15, 18, 19]
12 : [0, 2, 3, 5, 7, 11, 13, 17, 18, 19]
13 : [1, 2, 4, 5, 10, 12, 15, 16, 17, 18, 19]
14 : [0, 4, 5, 6, 7, 11, 15]
15 : [1, 3, 6, 9, 10, 11, 13, 14, 16, 19]
16 : [2, 3, 4, 7, 13, 15, 17, 18, 19]
17 : [0, 3, 4, 5, 9, 10, 12, 13, 16, 19]
18 : [0, 1, 3, 4, 5, 6, 7, 9, 11, 12, 13, 16, 19]
19 : [1, 3, 4, 5, 6, 7, 9, 11, 12, 13, 15, 16, 17, 18]
--------------------------------------
ALCANZABILIDAD
Nodo s = 0
Nodo t = 6

El candidato propuesto es:
[0, 12, 11, 17, 10, 6]


El candidato propuesto sí es solución
```

```
davidhernandez@Davids-MacBook-Pro Programa01 % python3 Alcanzabilidad.py
Gráfica generada con 16 vértices generada: aleatoriamente:

0 : [1, 3, 4, 5, 6, 8, 9, 10, 12, 15]
1 : [0, 3, 4, 5, 6, 7, 9, 10, 12]
2 : [3, 5, 6, 9, 11, 15]
3 : [0, 1, 2, 5, 7, 8, 9, 10, 12, 15]
4 : [0, 1, 5, 6, 7, 8, 9, 13, 14, 15]
5 : [0, 1, 2, 3, 4, 7, 10, 11, 12, 13, 14]
6 : [0, 1, 2, 4, 7, 10, 11, 12]
7 : [1, 3, 4, 5, 6, 9, 10, 11, 12, 14]
8 : [0, 3, 4, 9, 10, 11, 12, 14, 15]
9 : [0, 1, 2, 3, 4, 7, 8, 10, 12, 13, 14, 15]
10 : [0, 1, 3, 5, 6, 7, 8, 9, 11, 12, 14, 15]
11 : [2, 5, 6, 7, 8, 10, 12]
12 : [0, 1, 3, 5, 6, 7, 8, 9, 10, 11, 14, 15]
13 : [4, 5, 9, 14]
14 : [4, 5, 7, 8, 9, 10, 12, 13]
15 : [0, 2, 3, 4, 8, 9, 10, 12]
----------------------------------------
ALCANZABILIDAD
Nodo s = 6
Nodo t = 14

El candidato propuesto es:
[6, 1, 12, 9, 10, 0, 5, 4, 15, 2, 14]


El candidato propuesto sí es solución
```

```
davidhernandez@Davids-MacBook-Pro Programa01 % python3 Alcanzabilidad.py
Gráfica generada con 19 vértices generada: aleatoriamente:

0 : [3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16, 18]
1 : [2, 3, 4, 6, 7, 9, 10, 12, 13, 16]
2 : [1, 4, 7, 10, 11, 15, 17]
3 : [0, 1, 5, 6, 7, 9, 10, 12, 14, 15, 18]
4 : [0, 1, 2, 5, 8, 17]
5 : [0, 3, 4, 7, 8, 9, 10, 11, 12, 15, 16, 17, 18]
6 : [0, 1, 3, 7, 8, 9, 12, 14, 15, 16, 18]
7 : [0, 1, 2, 3, 5, 6, 14, 18]
8 : [0, 4, 5, 6, 9, 11, 12, 15, 16, 17]
9 : [0, 1, 3, 5, 6, 8, 10, 11, 14, 18]
10 : [0, 1, 2, 3, 5, 9, 11, 13, 16, 17, 18]
11 : [2, 5, 8, 9, 10, 12, 13, 16, 18]
12 : [0, 1, 3, 5, 6, 8, 11, 14, 15, 17]
13 : [1, 10, 11, 17]
14 : [0, 3, 6, 7, 9, 12, 17, 18]
15 : [2, 3, 5, 6, 8, 12, 16]
16 : [0, 1, 5, 6, 8, 10, 11, 15, 17, 18]
17 : [2, 4, 5, 8, 10, 12, 13, 14, 16, 18]
18 : [0, 3, 5, 6, 7, 9, 10, 11, 14, 16, 17]
----------------------------------------
ALCANZABILIDAD
Nodo s = 6
Nodo t = 4

El candidato propuesto es:
[6, 3, 5, 12, 14, 9, 11, 16, 1, 15, 4]


El candidato propuesto sí es solución
```

```
davidhernandez@Davids-MacBook-Pro Programa01 % python3 Alcanzabilidad.py
Gráfica generada con 17 vértices generada: aleatoriamente:

0 : [1, 2, 5, 6, 7, 10, 11, 13]
1 : [0, 5, 7, 9, 10, 11, 12, 13, 15, 16]
2 : [0, 3, 8, 9, 14, 16]
3 : [2, 6, 7, 11, 13, 16]
4 : [5, 6, 8, 9, 10, 12, 14]
5 : [0, 1, 4, 7, 10, 11, 12, 13, 15, 16]
6 : [0, 3, 4, 8, 12]
7 : [0, 1, 3, 5, 8, 12, 13, 16]
8 : [2, 4, 6, 7, 10, 13, 14, 16]
9 : [1, 2, 4, 10, 15]
10 : [0, 1, 4, 5, 8, 9, 14, 15]
11 : [0, 1, 3, 5, 14, 15]
12 : [1, 4, 5, 6, 7, 16]
13 : [0, 1, 3, 5, 7, 8, 14, 15]
14 : [2, 4, 8, 10, 11, 13]
15 : [1, 5, 9, 10, 11, 13, 16]
16 : [1, 2, 3, 5, 7, 8, 12, 15]
---------------------------------------
ALCANZABILIDAD
Nodo s = 7
Nodo t = 8

El candidato propuesto es:
[7, 5, 11, 14, 1, 15, 9, 4, 6, 12, 16, 13, 3, 0, 2]


El candidato propuesto no es solución
```

b) **3-SAT**.

- **Ejemplar Genérico** Un conjunto de clausulas C de longitud 3.

  Enunciado de Optimización Determinar la asignación de las variables de las clausulas tal que en su conjunto son verdaderas.

  Enunciado genérico Un conjunto de clausulas C de longitud 3 y una asignación A.

  Pregunta de decisión ¿Existe una asignación A para las variables de C tal que C es verdadera?

b) Diseñar un algoritmo no-determinístico polinomial.

Para este caso, es un poco más sencillo que el problema anterior.

En este caso por cada clausula se evalúa su valor de verdad y se va actualizando una variable global, la cual tendrá como resultado la evaluación final, esto se se hace evaluando cada uno de los valores de sus variables.
Por ejemplo si tenemos una clausula así:

$$[p,\ q, s]$$

Esto significa:

$$p \vee\ q \vee s$$

```python
def create_random_clauses(list_chars, num_clauses):
    aux_dict = {c : set() for c in list_chars}
    clauses = []

    val_variables = {}

    for _ in range(num_clauses):
        neg = random.choice(BOOLEAN_LIST)
        clauses.append(Clause([], neg))

    for i in range(len(clauses)):
        for _ in range(3):
            c = random.choice(list_chars)

            while i in aux_dict[c]:
                c = random.choice(list_chars)

            aux_dict[c].add(i)

            val = random.choice(BOOLEAN_LIST)
            neg = random.choice(BOOLEAN_LIST)

            if c in val_variables:
```

```
                clauses[i].add_variable(Variable(c, val_variables[c
    ], neg))
            else:
                clauses[i].add_variable(Variable(c, val, neg))
                val_variables[c] = val

    return clauses

def evaluate_3sat(clauses):
    evaluation = clauses[0].get_valor()

    for clause in clauses[1:]:
        evaluation &= clause.get_valor()

    return evaluation
```

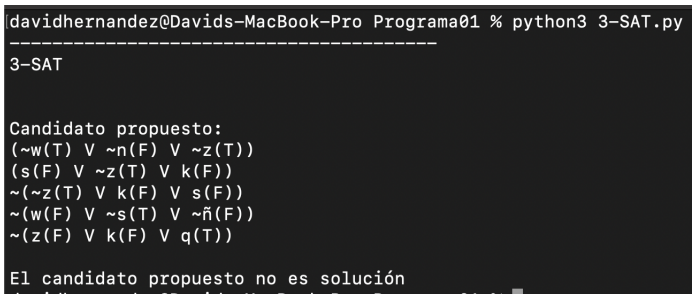
Listing 2: Algoritmo para 3-SAT

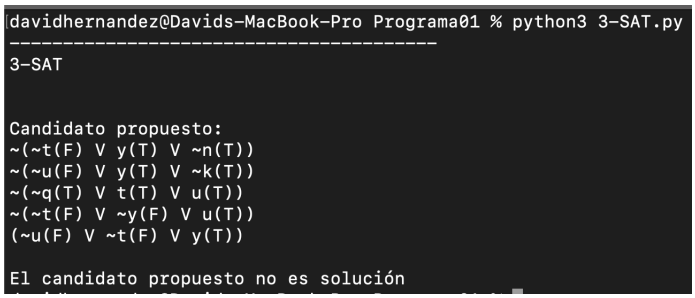
c) Implementar el algoritmo diseñado.

```
davidhernandez@Davids-MacBook-Pro Programa01 % python3 3-SAT.py
----------------------------------------
3-SAT


Candidato propuesto:
~(d(T) V ~c(T) V x(T))
(f(F) V d(T) V c(F))
(a(T) V x(T) V ~w(F))
(c(F) V f(F) V ~x(F))
~(~s(T) V w(T) V d(T))

El candidato propuesto no es solución
```

```
davidhernandez@Davids-MacBook-Pro Programa01 % python3 3-SAT.py
----------------------------------------
3-SAT


Candidato propuesto:
~(v(T) V ~f(T) V w(T))
~(~k(T) V v(T) V r(T))
(g(F) V ~r(F) V f(F))
~(v(T) V ~r(F) V ~j(F))
~(r(T) V ~f(T) V ~ñ(T))

El candidato propuesto no es solución
```

```
davidhernandez@Davids-MacBook-Pro Programa01 % python3 3-SAT.py
----------------------------------------
3-SAT


Candidato propuesto:
~(~s(T) V q(T) V ~h(F))
(q(T) V i(F) V ~t(T))
(i(F) V ~l(T) V f(F))
~(~z(T) V ~l(T) V ~g(T))
~(l(F) V ~i(T) V ~z(T))

El candidato propuesto no es solución
```


davidhernandez@Davids-MacBook-Pro Programa01 % python3 3-SAT.py
3-SAT
Candidato propuesto:
(~w(T) V ~n(F) V ~z(T))
(s(F) V ~z(T) V k(F))
~(~z(T) V k(F) V s(F))
~(w(F) V ~s(T) V ~ñ(F))
~(z(F) V k(F) V q(T))
El candidato propuesto no es solución


davidhernandez@Davids-MacBook-Pro Programa01 % python3 3-SAT.py
3-SAT
Candidato propuesto:
~(~t(F) V y(T) V ~n(T))
~(~u(F) V y(T) V ~k(T))
~(~q(T) V t(T) V u(T))
~(~t(F) V ~y(F) V u(T))
(~u(F) V ~t(F) V y(T))
El candidato propuesto no es solución