Clase Práctica 6

Ej 1:

1. Decidir cuáles de los siguientes problemas se pueden resolver en L o en NL.

a) • PALINDROME = $\{x : x$ es una cadena palíndroma$\}$

b) • DIRECTED_CYCLE = $\{\langle G\rangle : G$ es un grafo dirigido y contiene un ciclo$\}$

c) • BALANCED = $\{x : x$ tiene tantos 1s como 0s$\}$

d) • UNDIRECTED_CYCLE = $\{\langle G\rangle : G$ es un grafo no dirigido y contiene un ciclo $C\}$

a) PALINDROME = {x: x es una cadena palíndroma} $\in L$ → sos ; somos ; ama ; reconocer ; poop (?

Bueno, ¿cómo sería lo primero q' se me ocurre?

poop i++ y j-- por $\lfloor |s|/2 \rfloor$ pasos viendo q': s[i] == s[j].

Hago M máq. det. q' recibe un string s como entrada, de manera que:

M: ⟨s⟩

```
i := 0 ; j := |s| - 1                    O(2 log n)
for (k := 0; k < ⌊|s|/2⌋ ; k++)          O(log n)
    if (s[i] != s[j]):                   O(2 log n)
        ret false
    i++
    j--
ret true
```

δ

b) DIRECTED_CYCLE = {⟨G⟩: G es un grafo dirigido y contiene un ciclo} $\in NL$

Sea N máq. no-det en logspace tal que:

N: ⟨G⟩

```
for (i := 0; i < |V|; i++):              O(log |V|)        ← i es a lo sumo la rep. binaria del número |V|
    actual := i                          O(log |V|)
    for (m := 0; m < |V|; m++):          O(log |V|)        (A lo sumo mide n-1 edges el ciclo)
        z := Inventa un nodo de V.       O(log |V|)
        if ((actual, z) ∈ E):
            if (z == i):                 → Hay un ciclo
                ret true
            actual := z
ret false                                → Intentó todo y no hay un ciclo.
```

c) BALANCED = {x : x tiene tantos 1's como 0's} ∈ L

La cantidad de 1's y 0's en x son números representables en log |x| bits.

M: ⟨x⟩

c := 0

→ O(log|x|) pq' lo máximo es representar el nro de la longitud en binario.
→ Contador de 1's y 0's, si es 0 hay = cantidad, si es > 0, hay más 1's, si es < 0, hay más 0's.

→ i es a lo sumo la rep. binaria del nro |x|

for (i = 0; i < |x|; i++): O(log|x|)

  if (x[i] == 1):

    c++

  if (x[i] == 0):

    c--

ret (c == 0)

d) UNDIRECTED-CYCLE = {⟨G⟩ : G es un grafo no dirigido y contiene un ciclo C}

N: ⟨G⟩

  for (i = 0; i < |V|; i++):

    actual := i

    for (m = 0; m < |V|; m++):

      z := Inventa un nodo de V.

      if ((actual, z) ó (z, actual) ∈ E):

        if (z == i):
          ret true

        actual := z

  ret false

Es casi el mismo código q' (b) excepto por esto

Ej 2:

2. Probar que DIRECTED_CYCLE es NL-completo.

DIRECTED_CYCLE = {⟨G⟩ : G es un grafo dirigido y contiene un ciclo}

Que D-C ∈ NL fue visto en el punto 1.6.

D-C ∈ NL-hard:

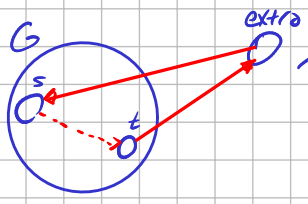
Para esto uso el hecho de que sé que PATH ∈ NL-Completo (más específicamente uso el q' sea NL-hard).

PATH $\leq_L$ D-C (Si puedo reducir un NL-hard a algún L ⊆ {0,1}* entonces L es NL-hard).

x ∈ PATH si: f(x) ∈ D-C, con f trabajo-L computable.

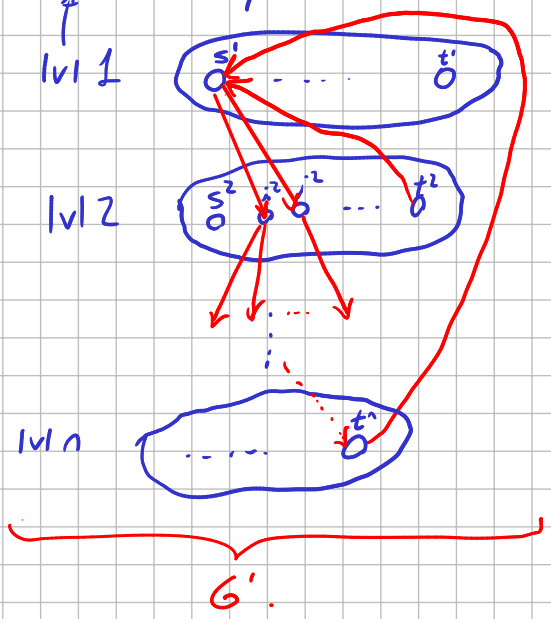
La f que voy a necesitar la voy a construir de la siguiente forma:

Quiero generar G' a partir de $\langle G, s, t\rangle$ q' es una X para PATH.



→ Me estoy imaginando algo así, pero, no me funciona si G ya tiene un ciclo desde antes.

Es de level, las cigs



¿Algo así?

Hago lo siguiente:

- Copio V n veces ($n = |V|$) para G' (son las n levels)
- Del lvl 1 uno s' a los nodos adyacentes de s en el grafo G, luego para el resto z^K del lvl K lo uno con todos los nodos adyacentes de z en el grafo G en el nivel K+1 (Esto es logarítmico de espacio, la única q' me guardo es el nodo al q' le busco los adyacentes y los voy poniendo en la salida).
- Por último uno los t's de todos los niveles con el s del primer nivel

¿Por qué esto funciona?

Es como estirar el grafo G, de manera q' la única forma de que haya un ciclo es si desde s' puedo llegar a un t^K con $K > 1$.

Ej 3:

3. Sea $L^2 = \mathsf{SPACE}[\log^2 n]$. Probar que $\mathsf{NL} \subseteq \mathsf{L}^2$.

Sea $L \in NL$, o sea, tengo N no-det tal q' $L(N)$ y N corre en espacio logspace.

Para computar N en una M det. q' corra en espacio $\log^2 n$ uso el hecho de que puedo guardar el recorrida que voy haciendo en el árbol de ejecución de N (pq' esto toma la altura del arbol y esto lo hago reutilizando el espacio)

Por cada nodo guardo todo el camino ya hecho, escribiendo a cada izq. o hijo único y 1 como der.

$\log n$

(Es el Algoritmo de Savitch esto)

Otra forma (más sencilla y aplicada)

Aplicar el teorema de Savitch q' dice q'

$$NSpace(S(n)) \subseteq Space(S(n)^2)$$

con S una función tiempo construible, si $S(n) = \log n \Rightarrow NL \subseteq L^2$

Ej 4:

4. Probar que PH $\subseteq$ PSPACE.

Idea:
Para ver esto tengo q' usar un $L \in PH$ cualquiera y ver que pertenezca a PSPACE.

Sea $L \in PH$, o sea, $L \in \Sigma_i^p$ (ya que el lenguaje se encuentra en alguna parte de $\bigcup_k \Sigma_k^p$).

Tengo M máq. det. q' corre en tiempo poly. y p un polinomio tal q':

$$x \in L \text{ sii } \exists u_1 \in \{0,1\}^{p(|x|)} \forall u_2 \in \{0,1\}^{p(|x|)} \dots Q_i u_i \in \{0,1\}^{p(|x|)} . M(x, u_1, \dots, u_i) = 1$$

Luego, asumo q' Q es un $\exists$ (asumo q' i es impar por conveniencia, pero la demo. es análoga si: "Q es un $\forall$").

Entonces,

$$x \in L \text{ sii } \exists u_1 \forall u_2 \dots \exists u_i . M(x, u_1, u_2, \dots, u_i) = 1.$$

Para demostrar q' esto está en PSPACE debo hacer una máq. det. q' corre en espacio poly, a esta máquina la llamo M' y va a hacer lo siguiente:

```
M': ⟨x⟩
  M''(x, i, ∅)

M'': ⟨x, k, C⟩      ← Máq. recursiva para ver los ∃ y ∀ de los certificados
                      C es el cjto de certificados ya obtenidos, k la cantidad de
                      certificados restantes y x la entrada.
  if (k == 0):
    ret M(x, c1, c2, ..., c|c|)
  if (|C| % 2 == 0):        Si el índice es par sigue un ∃.
    ret M'(x, k-1, C ∪ {a}) ∨ ... ∨ M'(x, k-1, C ∪ Z)
                            donde Z es un cjto con tamaño ≤ p(|x|)
  if (|C| % 2 == 1):
    ret ⋀_{0 ≤ l ≤ p(|x|)} M'(x, k-1, Z_l)
```

Todas las posibles combinaciones de cjtos con tamaño menor o igual a $p(|x|)$.

Como cada Z_l lo computo en el mismo espacio, este toma $p(|x|)$ nada más, la cúal es poly, y como M computa en tiempo poly, no puede llegar a usar más q' un espacio poly.

✓ 11

Ej 5:

5. ¿A qué pisos pertencen los siguiente problemas?

a) • $\exists\exists!sat = \{\langle \varphi(x,y) \rangle : \varphi$ es una fórmula 3-CNF y existe una asignación de las variables x tal que hay una única asignación de la variable y que satisface $\varphi\}$

b) • 3-COLORING-EXTENSION $= \{\langle G, W \rangle : G$ es un grafo y W es un subconjunto de sus nodos tal que todo 2-coloreo de los nodos de W puede extenderse a un coloreo de $G\}$

a)

Buena, vamos a parafrasear lo q' nos dice el lenguaje:

- Que exista un único y implica que existe un y tal q' para cualquier otra asignación de y la fórmula φ no sea satisfacible.

Como no sé cuales son las variables de x y la y tengo que tenerlo en cuenta ya.

$\exists\exists!Sat$ s:: $\exists M$ máq. det. poly que vea que $\varphi(x,y) \in \exists\exists!Sat$.

s:: $\exists \langle x, y_0 \rangle \forall y \; (\varphi(x, y_0) \wedge (y \neq y_0 \Rightarrow \neg\varphi(x, y))$

Asignación de las variables

Otra asignación de la misma variable q' y_0.

Se puede resolver por una máq. M det. poly.

s:: $\exists\exists!Sat \in \Sigma_2^P$ ✓

b)

Para todo 2-Coloreo de W (q' es un subcjto de nodos de G) se puede extender a un 3-coloreo de G.

$\forall W_v \exists G_v . ($ W_v es 2-coloreo de W $\Rightarrow$ G_v es 3-coloreo de G (vea q' no hayan 2 nodos adyacentes con el mismo color) y $G_v|_W = W_v$ (G_v restringido a los nodos de W tienen la asignación de W_v)

W_v := 2-coloreo de W
G_v := 3-coloreo de G. ✓

Ej 6:

> 6. Probar que $\exists\exists!sat$ es hard para su clase.

$\Sigma_2 sat = \{\langle \varphi(x,y) \rangle : \exists x_v \forall y_v . \; F\varphi(x_v, y_v)\}$

Sé que para todo $i>0$, $\Sigma_i SAT \in \Sigma_i^P$-hard.

Luego, qvq:

$\Sigma_2 sat \leq_P \exists\exists!sat$

Veo de parafrasear la def. de $\Sigma_2 sat$

$\Sigma_2 sat$ s:: $\exists x \forall y . \varphi(x,y)$

s:: $\exists x \neg\exists y . \neg\varphi(x,y)$

Luego, a $\neg\exists y . \neg\varphi(x,y)$ lo puedo reescribir de la siguiente forma:

$\exists \mathbf{x} \exists! \mathbf{y}, \mathbf{y} : (y \vee \neg\phi(\mathbf{x}, \mathbf{y})) \wedge (\bar{y} \vee y_1) \wedge \cdots \wedge (\bar{y} \vee y_m),$ $= \psi$

Esto sale de que: $Y = Y_1 Y_2 \ldots Y_n$

It is easy to show that USAT is coNP-hard: a formula $\phi(x_1, \ldots, x_n)$ is unsatisfiable if and only if

negación de x.

$$(x \vee \phi(x_1, \ldots, x_n)) \wedge (\bar{x} \vee x_1) \wedge \cdots \wedge (\bar{x} \vee x_n) \tag{1}$$

has exactly one satisfying assignment (namely, the assignment where every variable is true). USAT is not known to be NP-hard, but Valiant and Vazirani

Yo tengo que $\neg \exists y. \neg\varphi(x,y)$, o sea no existe y tal que la fórmula $\neg\varphi(x,y)$ es satisfacible, por ende digo que $\neg\varphi(x,y) \in USAT$, o sea que para obtener una fórmula con única solución tengo que tener una variable y, tal que cada elem. de este vaya como parámetro de la fórmula $\neg\varphi$ y el resto esté en una conjunción tal q' cada elemento sea una disyunción de y_i y $\bar{y}$.

Si $\neg\varphi$ fuese satisfacible existiría más de una valuación q' la satisface a la nueva fórmula ψ, si no la es, la única q' existe es que, todas las variables sean 1.

Con esto (y viendo q' también se puede pasar a 3CNF en tiempo poly) obtengo una f computable polinómialmente tal que

$$x \in \Sigma_2 SAT \iff f(x) \in \exists!SAT$$

$$f = \varphi(x,y) \mapsto \psi(x,y)$$

γ