Pregunta 1

1.

Se tiene un algoritmo A de caja negra de resolución SAT, es decir, un dispositivo que toma una fórmula de lógica proposicional ϕ, A(ϕ) es verdadero si ϕ es satisfacible.

a) Algoritmo que utiliza A como subrutina para determinar si ϕ es una tautología:

```
def esTautologia(p):
    if A(~p) == 0: #Si ~p es insatisfacible
        return True
    else:
        return False
```

Probaremos si el algoritmo anterior es correcto:

Demostración. PDQ: $\neg\phi$ es insatisfactible $\Leftrightarrow$ ϕ es tautología

$$\begin{aligned}\neg\phi \text{ es insatisfactible} &\Leftrightarrow (\emptyset \cup \{\neg\phi\}) \text{ es insatisfactible}\\ &\Leftrightarrow \emptyset \models \phi\\ &\Leftrightarrow \sigma(\phi) = 1\\ &\Leftrightarrow \phi \text{ es Tautología}\end{aligned}$$

□

b) Se tienen 2 fórmulas proposicionales ϕ y φ, se va a determinar si tienen los mismos valores de verdad. Algoritmo que utiliza A para responder a esta pregunta:

```
def equivalentes(p, q):
    if A(~((~p or q) and (p or ~q))) == 0:
        return True
    else:
        return False
```

Probaremos si el algoritmo anterior es correcto:

Demostración. PDQ: $\neg((\neg\phi \vee \varphi) \wedge (\phi \vee \neg\varphi))$ es insatisfactible $\Leftrightarrow \phi \equiv \varphi$

$$\begin{aligned}\neg((\neg\phi \vee \varphi) \wedge (\phi \vee \neg\varphi)) \text{ es insatisfactible} &\Leftrightarrow (\neg\phi \vee \varphi) \wedge (\phi \vee \neg\varphi) \text{ es Tautología}\\ &\Leftrightarrow (\phi \Rightarrow \varphi) \wedge (\varphi \Rightarrow \phi) \text{ es Tautología}\\ &\Leftrightarrow (\phi \Leftrightarrow \varphi) \text{ es Tautología}\\ &\Leftrightarrow \sigma(\phi) = \sigma(\varphi)\\ &\Leftrightarrow \phi \equiv \varphi\end{aligned}$$

□

c) Se tiene una fórmula proposicional ϕ con n variables que se sabe que es satisfacible. Algoritmo que utiliza A como subrutina para obtener una asignación satisfactoria para ϕ utilizando como máximo n llamadas a A:

Sean $\phi_1, ..., \phi_n$ proposiciones de la fórmula, luego sea ϕ' tal que $\sigma(\phi_1) = 1$

Si $A(\phi')$ satisfacible $\Rightarrow \sigma(\phi_1) = 1$
Si $A(\phi')$ insatisfacible $\Rightarrow \sigma(\phi_1) = 0$

Luego ϕ'' tal que $\sigma(\phi_1) = \{\text{valor con el que es satisfacible}\}$ y $\sigma(\phi_2) = 1$, se repite el proceso anterior n veces hasta encontrar todas las valuaciones $\sigma(\phi_k)$, $k \in 1, ..., n$.

Demostración. Para probar la correctitud se procederá por el principio de inducción:
$\boxed{\text{Caso Base:}}$
Sea ϕ una formula lógica con ϕ_1 su única proposición y se sabe que existe $\sigma(\phi_1)$ tal que $\sigma(\phi) = 1$. Sea entonces ϕ' tal que $\sigma(\phi_1) = 1$, como la fórmula depende solo de ϕ_1, si con $\sigma(\phi) = 1$ se cumple que $A(\phi')$ entrega satisfacible también se cumple que $\sigma(\phi') = 1$, por el contrario si $A(\phi')$ es insatisfacible entonces basta con tomar $\sigma(\phi) = 0$, con esa valuación se tendría el otro caso.
$\boxed{\text{Caso Inductivo:}}$
Supongamos que existen n-1 valuaciones $\sigma(\phi_1), ..., \sigma(\phi_{n-1})$ con las que se cumple $A(\phi^{(n-1)})$ entrega satisfacible. Como se sabe que $\phi^{(n-1)}$ es satisfacible entonces debe existir una valuación $\sigma(\phi_n)$ tal que $\sigma(\phi^{(n-1)}) = 1$. Entonces para encontrar la n-ésima valuación basta con tomar $\phi^{(n)}$ tal que $\sigma(\phi_n) = 1$, si con esa valuación $A(\phi^{(n)})$ entrega que es satisfacible entonces encontramos todas las valuaciones con las que $\sigma(\phi) = 1$. En caso de que $A(\phi_{(n)})$ entregue insatisfacible, basta tomar $\sigma(\phi_n) = 0$.

□

El algoritmo anterior nos entrega las n valuaciones de las variables de la formula proposicional con solo n llamadas.

2.

Notemos que $\sigma(a_i) = \sigma(b_i)$ es un factor importante al momento de sumar los números binarios, por lo que necesitamos definir una función auxiliar que indique si esto se cumple o no. Esto se consigue directamente de reescribir y manipular a_i XOR b_i:

$$iguales_i = \neg(\neg a_i \vee \neg b_i) \vee \neg(a_i \vee b_i)$$

Donde $\sigma(iguales_i) = 1$ si y solo si $\sigma(a_i) = \sigma(b_i)$.

Otro factor importante es cuando ocurre lo que denominamos 'arrastre', lo cual ocurre solo de las siguentes formas para este caso: $(1 + 1 = 0$ con arrastre) y $(1 + 1 + 1 = 1$ con arrastre). Lo que hace el arrastre, en esencia, es que si ocurren una de las sumas anteriormente descritas entonces a la siguiente posición se la suma un 1 extra. Definimos lógicamente si es que se 'arrastró' o no un 1 desde la posición i-1 hasta la posición i como la función auxiliar siguente:

$$arrastre_i = \neg(\phi_{i-1} \vee iguales_{i-1}) \vee \neg(\neg a_{i-1} \vee \neg b_{i-1})$$

Donde $\sigma(arrastre_i) = 1$ si y solo si se arrastró un 1 desde la posición i-1 hacia la posición i.

Dadas las definiciones anteriores, construimos nuestras fórmulas ϕ_i, distinguendo los siguentes casos:

$\boxed{\text{Caso inicial}}$ (i=0) Para este caso, $\sigma(\phi_i) = 1$ ssi $\sigma(a_i) \neq \sigma(b_i)$

$$\Rightarrow \phi_i = \neg iguales_i \ , \ donde \ i = 0$$

$\boxed{\text{Caso intermedio}}$ $(i \in \{1, ..., n-1\})$ (Dado $n \geq 2$) Necesitamos que $\sigma(\phi_i) = 1$ ssi $\sigma(a_i) \neq \sigma(b_i)$ y que no haya ocurrido algun arrastre hacia la posición i, o que sí haya ocurrido arrastre hacia la posición i, pero que se cumpla que $\sigma(a_i) = \sigma(b_i)$. Es decir:

$$\phi_i = \neg(iguales_i \vee arrastre_i) \vee \neg(\neg arrastre_i \vee \neg iguales_i) \ , \ donde \ i \in \{1, ..., n-1\}$$

$\boxed{\text{Caso final}}$ (i=n): No existen a_n ni b_n para compararlos entre ellos, y notamos que $\phi_{i=n}$ depende unicamente del arrastre que haya ocurrido desde la posición anterior, de la siguiente forma:

$$\phi_i = arrastre_i \ , \ donde \ i = n$$

Con esto se han construido, para todo $i \in \{0, 1, ..., n-1, n\}$, los ϕ_i tales que: $\sigma(\phi_i) = 1$ ssi el i-ésimo bit de la suma es un 1. Notemos que las construcciones fueron expresadas solo a base de los operadores $\{\neg, \vee\}$ (incluyendo las funciones auxiliares, las cuales también fueron definidas solo a base de los operadores indicados), con lo que se cumple con la restricción dada.