Trabajo Práctico n.º 3 - Teoría de Algoritmos

Cuatrimestre	1
Año	2022
Estudiantes	Aldana Rastrelli (98.408) Guido Ernesto Bergman (104.030) Iván Loyarte (97.213) Ramiro Andrés Fernández Márquez(105.595) Santiago Ronchi (101.043)
Fecha de Entrega	15 de junio de 2022
N° de entrega	2da
Grupo	Grupo 45

▼ Índice

Parte 1: El viaje a Qatar

1.
    - Min Cost Max Flow
    - Cycle Cancelling Algorithm
    - Successive Shortest Path Algorithm
2.
    - Pseudocódigo
    - Análisis de Complejidad Espacial
    - Análisis de Complejidad Temporal
    - Optimalidad
3.
4.
5.

Parte 2: Un reality único

1.

Explicación Exact Cover

- Demostración de que un problema es NP-C

Demostración de que el problema Casting es NP

Demostración de que el problema Casting es NP-Hard

- T1: Transformar la entrada del Exact cover en la del Casting
- T1’: Transformar la solución del Casting en la solución del Exact cover

2.

Demostración de que el Exact cover es NP

Demostración de que el Exact cover es NP-Hard

Transformaciones

- T1: Transformar la entrada del SAT en la del SAT usando hasta 3 literales por clausula
- T1’: Transformar la solución del SAT usando hasta 3 literales por clausula en la del SAT
- T2: Transformar la entrada del SAT usando hasta 3 literales por clausula en la del Chromatic number
- T2’: Transformar la solución del Chromatic number en la del SAT usando hasta 3 literales por clausula
- T3: Transformar la entrada del Chromatic number en la del Exact cover
- T3’: Transformar la solución del Exact cover en la del Chromatic number

3.
4.
5.

Correcciones primera entrega

Parte 1: El viaje a Qatar

💡 Una ONG con sede en Buenos Aires desea realizar un viaje grupal de “estudio” a Qatar entre las fechas de 21 de noviembre de 2022 y el 18 de diciembre de 2022. Han realizado diversas averiguaciones con compañías aéreas para conocer el costo de pasaje y la cantidad que podrían comprar para diferentes trayectos por ciudades del mundo. Su objetivo es determinar cuál es la máxima cantidad de personas que podría viajar y hacerlo al menor costo posible.

▼ **Formato de los archivos**

El programa debe recibir por parámetro el path del archivo donde se encuentra el grafo.
El formato del archivo es de texto. Las primeras dos líneas corresponden al nodo fuente y sumidero respectivamente. Continúa con una línea por cada eje del grafo con el formato: ORIGEN,DESTINO,COSTO UNITARIO,CAPACIDAD.

El programa debe retornar en pantalla la cantidad máxima de personas que pueden viajar y el costo mínimo que se puede gastar.

1.

💡 Investigar y seleccionar uno de los siguientes algoritmos que resuelven este problema conocido como flujo máximo con costo mínimo (“Min Cost Max Flow”): “Cycle Cancelling Algorithm” o “Successive shortest path algorithm”.

Min Cost Max Flow

Encontrar el flujo máximo de costo mínimo de una red es un problema equivalente a encontrar la circulación de costo mínimo.

Cycle Cancelling Algorithm

Para comenzar con se utiliza cualquier algoritmo de flujo máximo para establecer un posible flujo en el grafo, luego se intenta mejorar esto encontrando ciclos negativos en la red residual. La cantidad de iteraciones que realiza este algoritmo (gracias a la suposición de que una respuesta EXISTE) será un número de x veces basadas en la cantidad de datos Ax que tenga el grafo.

Successive Shortest Path Algorithm

Se centra en resolver la problemática del “max-flow, min-cost” utilizando la siguiente idea:

En lugar de buscar el flujo máximo, usaremos $Bellman - Ford$ para el primer camino de menor costo de s (ciudad de origen) a t (ciudad destino). Teniendo esto empezaremos a construir el grafo residual viendo de $n(i, j)$ el cual será $n(res(i, j))$ y le restamos la capacidad del grafo inicial. Repetimos el proceso hasta que no queden caminos desde s hasta t, y así sabremos que el flujo es máximo. Como sabemos que este grafo es ahora maximal, se corresponde con una posible respuesta del problema original de $min - cost \ \ max - flow$ y esta será óptima.

Este método es aplicable cuando nuestro grafo original no tiene ciclos negativos, de lo contrario sería matemáticamente imposible determinar un “costo mínimo”. Con el grafo residual ya armado, podemos utilizar una vez más $Bellman - Ford$ para obtener nuestro resultado final.

➡️ Decidimos seleccionar el algoritmo $Cycle \ Cancelling \ Algorithm$

2.

💡 Explicar cómo funciona el algoritmo seleccionado. Incluir:
pseudocódigo, análisis de complejidad espacial, temporal y optimalidad.

Seleccionaremos un nodo `s` como `source` y otro nodo `t` como `sink`. En el problema a resolver, `s` será la ciudad de origen y `t` la ciudad destino.

Calcula el flujo máximo posible ignorando la información de costos.

Mientras puedan existir ciclos negativos, entra en el circuito principal:

El ciclo principal identifica repetidamente ciclos negativos en el gráfico residual.

Al ejecutar el algoritmo de **Bellman-Ford**, buscamos un ciclo de costo negativo en el gráfico residual.

El ciclo negativo se elimina del gráfico residual saturando uno de los bordes.

Pseudocódigo

```
Sea G el grafo
Sea s el source y t el sink

s = elegir_nodo_de(G)
t = elegir_nodo_de(G)
```

```
Calcular flujo máximo (usaremos Ford Fulkerson)

WHILE aún puedan existir ciclos negativos:

  Ejecutar BELLMAN-FORD para encontrar ciclo negativo.

  IF ciclo negativo existe:

    identificar las aristas del ciclo
    ajuste = min( costo[e] | de todas las e aristas del ciclo)

    FOR e en las aristas del ciclo:
      flujo[e] += ajuste
  ELSE break

FIN
```

Análisis de Complejidad Espacial

El grafo ocupa un espacio $O(V \times E)$, siendo V la cantidad de vértices y E de las aristas.

Para Bellman-Ford, el tamaño de las estructuras de datos utilizadas (tomando como referencia la implementación realizada en el Trabajo Práctico 2 que usaba hash_aristas_min, distancias y ciclo) es, en el peor de los casos, proporcional a la cantidad de nodos que tiene el grafo. Es decir, $O(V)$.

No se usan otras estructuras de datos además de las mencionadas, por lo que la complejidad espacial final es de $O(V \times E)$.

Análisis de Complejidad Temporal

La complejidad del $Ford - Fulkerson$ se puede expresar como $O(E \times C)$, siendo C la suma de las capacidades de las aristas que salen de la ciudad de origen. Esto se debe a que:

- En cada iteración, Ford-Fulkerson aumenta el flujo en al menos una unidad y el flujo máximo no puede ser mayor que la suma de las capacidades de las aristas que salen de la ciudad de origen. Dado que esta suma es una característica de los elementos de entrada y no un parámetro del problema, el algoritmo es pseudopolinomial.
- Cada iteración de Ford-Fulkerson necesita $O(E)$ tiempo para encontrar una ruta de aumento, ya que el grafo residual tiene como mínimo E aristas y máximo $2E$ aristas, así que el tiempo para crearlo es $O(V + 2E) = O(E + E) = O(E)$.
- Actualizar el grafo residual tiene un valor de $O(V + E) = O(E)$, ya que se recorren todos los nodos y aristas del grafo original.

La cancelación de los ciclos negativos del grafo residual, tiene una complejidad de $O(E \times V \times S)$, siendo S la suma de todos los costos de los vuelos. Esto se debe a que:

- Cada iteración reduce el costo en al menos 1, por lo que la cantidad de iteraciones necesarias será como máximo la suma todos los costos. Dado que esta suma no es un parámetro del problema, el algoritmo es pseudopolinomial.
- Cada iteración requiere una ejecución de $Bellman - Ford$ en $O(E \times V)$ y actualizar el grafo residual, que al igual que el caso anterior se puede hacer en $O(E)$.

Por lo que la complejidad temporal final es de $O(max(E \times C, E \times V \times S))$.

Optimalidad

Optimalidad de Ford-Fulkerson: el algoritmo termina cuando no hay caminos simples $s - t$ en el grafo residual. Esto implica que hay un corte (A, B) y que:

- Todas las aristas que van de A hacia B se encuentran saturadas, ya que si no lo estuvieran, habría un camino $s - t$ en el grafo residual.
- Todas las aristas que van de B hacia A tienen flujo 0. Si esto no ocurriera, habría un *backward edge* en el grafo residual creando un camino entre S y T.

Juntando ambas consecuencias, podemos ver que el flujo que sale de A es igual a la capacidad del corte encontrado. Dado el flujo máximo es igual a la capacidad del corte mínimo, ya que en algún momento el flujo tiene que pasar de A a B, y que no puede haber un corte con menos capacidad que el encontrado, porque sino no podría existir un flujo con ese valor, se puede observar que el flujo encontrado por el algoritmo de Ford-Fulkerson es el máximo.

Optimalidad del Cycle Canceling Algorithm: El teorema Negative Cycle Optimality establece que: Una solución factible x* es una solución óptima del problema de flujo de costo mínimo si y solo si el grafo residual $Gx*$ no contiene un ciclo dirigido de costo negativo. Esto implica que cuando el algoritmo finaliza, es decir cuando no hay ciclos negativos en el grafo residual, el costo obtenido es el mínimo.

Demostración de que no si el grafo residual $Gx*$ no contiene un ciclo dirigido de costo negativo entonces x es una solución óptima del problema de flujo de costo mínimo:*

Sea x una solución factible cualquiera, entonces x = x + los flujos de como máximo m ciclos dirigidos en $Gx*$ (Augmenting Cycle Theorem). Si $Gx*$ no contiene ciclos negativos, entonces el costo de x es al menos el costo de x*, por lo tanto x* es una solución óptima para el problema.*

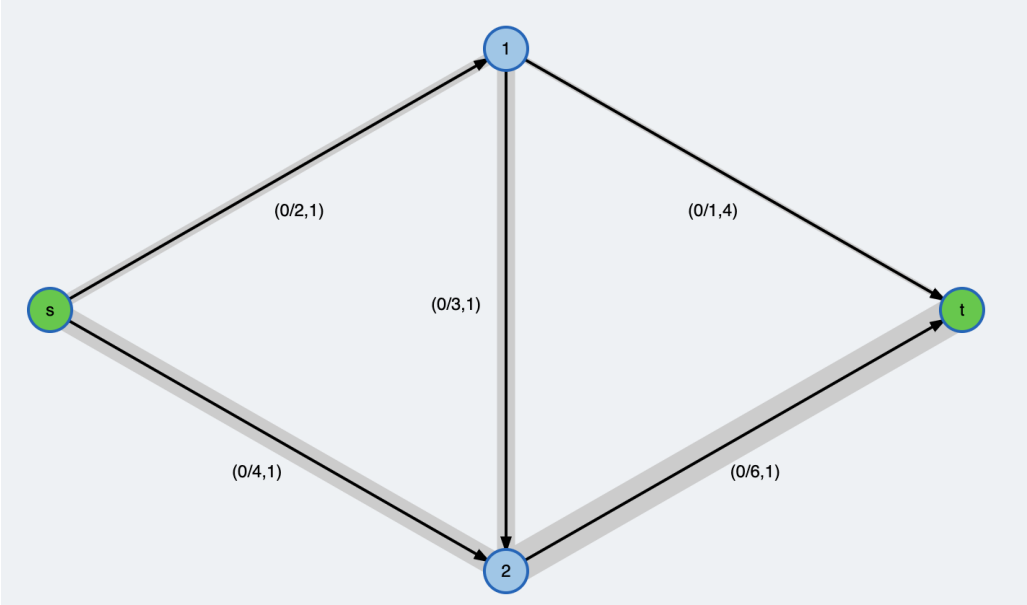
Existen múltiples algoritmos más avanzados para el problema de flujo de costo mínimo que logran mejores tiempos de ejecución. Una adaptación del algoritmo de cancelación de ciclo es la cancelación de ciclo medio mínimo, que da como resultado un algoritmo fuertemente polinomial.

💬 Algunas referencias
https://dspace.mit.edu/bitstream/handle/1721.1/2630/SWP-3914-35650575.p
https://www-di.inf.puc-rio.br/~laber/MinCostFlow.pdf

3.

💡 Dar un ejemplo paso a paso de su funcionamiento.

⟹ Se usan `source` y `target` que serán las ciudades de origen y destino respectivamente. Los resultados obtenidos van a estar directamente relacionados con los nodos usados como `source` y `target`, ya que lo que se va a buscar es el flujo máximo con el costo mínimo desde el `source` hasta el `target`.



Grafo ejemplo con `source` y `target` seleccionados (origen y destino).

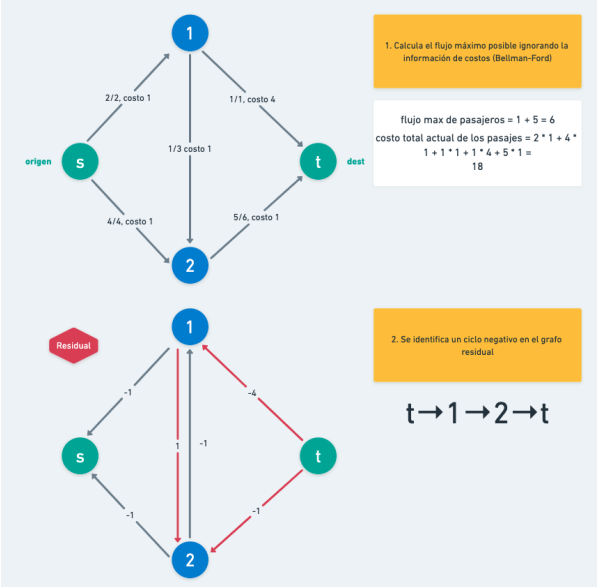


Imagen 1/3

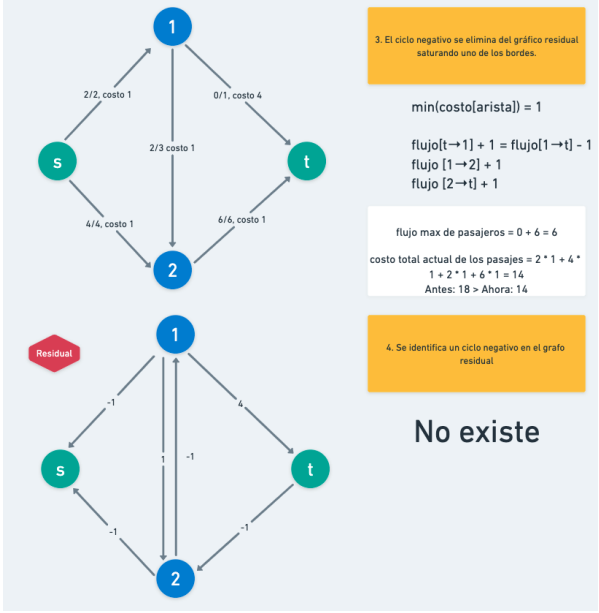


Imagen 2/3

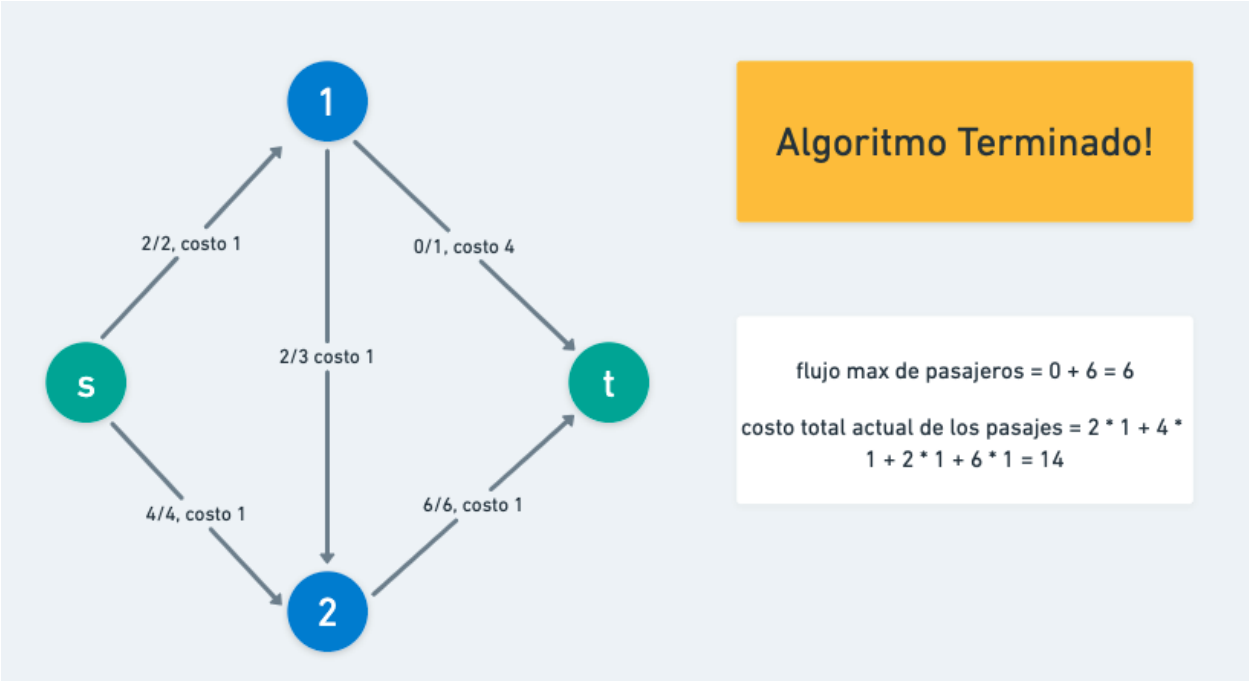


Imagen 3/3

4.

💡 Programar el algoritmo.

Para correr el algoritmo, se debe colocar en consola:

```
>> python3 main.py test.txt
```

5.

> 💡 Responder justificando: ¿La complejidad de su algoritmo es igual a la presentada en forma teórica?

La complejidad del algoritmo difiere de la presentada en forma teórica, ya que para encontrar los caminos de aumento se utilizó BFS. Esto implica que lo que se implementó es una variante de Ford Fulkerson, conocida como *Edmonds-Karp*. Esta variante presenta una complejidad $O(V * E^2)$.

Para demostrar esto, usaremos Gf: Grafo residual, y $\delta f(u, v)$ para la distancia más corta desde u a v en Gf, donde cada eje tiene distancia unitaria.

Si el algoritmo es usado en un grafo $G = (V, E)$ son nodos s y t, que en el problema a resolver representan las ciudades de origen y destino⇒ para todo nodo del grafo salvo $s, t:\ \delta f(u, v)$ en el grafo residual Gf incrementa monotonicamente con cada aumento del flujo.

Supongamos entonces para algún nodo v que podamos encontrar una optimización f que reduzca $\delta f(s, v)$ a $\delta f'(s, v) \Rightarrow \delta f'(s, v) < \delta f(s, v)$, es decir que la distancia disminuya. Eso implica que camino más corto de *s* a *v* en Gf' pase por *u*, tal que $(u, v) \epsilon . Ef$' y, por tanto, $\delta f'(s, u) = \delta f'(s, v) - 1$.

Pero esto sería un absurdo por nuestra elección previa de v y sabemos que:

$\delta f'(s, u) \geq \delta f(s, v)$.

El algoritmo siempre aumentará el flujo por los caminos más cortos entre nodos del grafo, y si tuviésemos un $(u, v) \epsilon Ef$, entonces no estaríamos cumpliendo nuestra hipótesis $\delta f'(s, v) < \delta f(s, v)$. Ergo podemos concluir que nuestro nodo v no existe.

Cuando se halla una arista crítica (una que cuando se aumenta su capacidad residual p es tal que $cf(p) = cf(u, v)$) esta misma desaparecerá del grafo residual cuando se haya aumentado el flujo en un camino, y por definición sabemos que al menos 1 arista debe ser crítica en cada camino. Por esto se deriva que toda arista de $|E|$ será crítica un máximo de $|V|/2$ veces.

Sean u, v nodos conectados del grafo, cuando esta arista sea crítica tendremos:

$\delta f(s, v) = \delta f(s, u) + 1$. Una vez se aumente el flujo, esta arista desaparecerá hasta que el camino inverso aparezca (si es que lo hace) en otro camino de aumento, entonces:

$$\begin{aligned} \delta f'(s, u) &= \delta f'(s, v) + 1 \\ &\geq \delta f(s, v) + 1 \end{aligned}$$

$$= \delta f(s, u) + 2$$

Podemos ver así que la distancia de u crece desde un mínimo de 0 a un mínimo de 2. Consecuentemente por la definición de un camino aumentado (u, v) los nodos en el camino mas corto no contendrán a s, u o t. Hasta no ser bloqueado de la fuente, la distancia sera como max $|V| - 2$ si es que esto ocurre. Con esto hacemos la cuenta:

$(|V| - 2)/2 = (|V|/2) - 1$ lo cual dará un total de $|V|/2$ veces que puede ser crítico. extrapolando esto a las $O(E)$ aristas del grafo nos da un total de $O(V * E)$.

Dado que la cantidad de iteraciones es $O(V * E)$ y cada iteración de Ford-Fulkerson tiene una complejidad de $O(E)$, por ejemplo para obtener el grafo residual, la complejidad final del *Edmonds-Karp* será $O(V * E^2)$

Dentro de las funciones internas que utilizamos para la cancelación de ciclos, se encuentran:

- Obtener el grafo residual nos implica una complejidad de $O(E)$.
- Para encontrar un ciclo negativo, utilizamos Bellman-Ford, que tiene una complejidad de $O(E \times V)$, por lo que es igual a su forma teórica.
- El armado del ciclo negativo (postorder) tiene una complejidad de a lo sumo $O(V)$. y el resto de las funciones utilizadas son de a lo sumo $O(E)$ u $O(V)$.

Por lo tanto, podemos decir que, como:

- El algoritmo de BFS respeta la complejidad de su forma teórica
- El algoritmo de Bellman-Ford respeta su complejidad de forma teórica
- El resto de las funciones utilizadas son a lo sumo $O(E)$ u $O(V)$.

La complejidad de la cancelación de los ciclos negativos sigue siendo $O(E \times V \times S)$, siendo S la suma de todos los costos de los vuelos. Esto se debe a que realiza iteraciones de Bellman − Ford $O(E \times V)$ a lo sumo una S cantidad de veces, en todas realizando una reducción del costo en al menos 1 unidad.

En conclusión, el $Cycle\ Cancelling\ Algorithm$ implementado no respeta la complejidad de su forma teórica, con un tiempo de ejecución $O(max(V * E^2, E \times V \times S))$. Este tiempo de ejecución sigue siendo pseudopolinomial, dado que depende de S que no es un parámetro del problema.

Parte 2: Un reality único

💡 Para un casting para un nuevo reality show han generado un conjunto de “k” características que desean que tengan los diferentes participantes. Por ejemplo: “historia trágica”, “habilidades musicales”, “capacidad atlética”, “estudios universitarios”, “amor por los animales”, etc. Cuentan con un conjunto de “n” personas que se anotaron con deseos de participar. Para cada característica tienen la lista de personas que la posee. La producción desea seleccionar a un subconjunto de participantes de forma tal de que cada una de las características se vea representada. Además para lograr mayor variabilidad quieren que no existan dos personas con la misma característica.

1.

💡 Utilizando EXACT-COVER demostrar que el problema al que denominaremos “casting” es NP-C

Explicación Exact Cover

El objetivo del Exact cover es encontrar en una colección S de subconjuntos de un conjunto X una subcolección S' que contenga cada elemento de X en exactamente un subconjunto de S'.

Ejemplo:

Sea $S = \{A, B, C, D\}$ una colección de subconjunto del conjunto $X = \{1, 2, 3, 4\}$ y:

- $A = \{\}$
- $B = \{1, 4\}$
- $C = \{2, 3\}$
- $E = \{3, 4\}$

En este caso, la subcolección $S' = \{B, C\}$ es una cobertura exacta de X debido a que los subconjuntos $B = \{1, 4\}$ y $C = \{2, 3\}$ son disjuntos y además $B \cup C = X$. Podríamos incluir el subconjunto A a S' y seguiríamos teniendo cobertura exacta (no así con el subconjunto E).

Supongamos también un conjunto $Y = \{1, 2, 3, 4, 5\}$. Para este conjunto no tenemos cobertura exacta utilizando S, debido a que ningún subconjunto de S contiene el 5.

Otro ejemplo conocido y de la vida cotidiana del problema de cobertura exacta es el de resolver un Sudoku.

Demostración de que un problema es NP-C

Un problema C es NP-C cuando:

- El algoritmo de certificación de C puede verificar una posible solución de cualquer instancia del problema en tiempo aceptable (polinomial). Esto quiere decir que C está contenido en el conjunto de problemas NP.

- Todo problema perteneciente a NP puede ser reducido polinomialmente a c (Supongamos $Y \subseteq NP / Y \leq_p C$). Esto implica que el problema C es NP-Hard.

Para demostrar que un problema C es NP-C es necesario:

- Demostrar que C tiene un algoritmo polinomial de certificación, es decir que es NP.
- Demostrar que es NP-Hard. Para esto alcanza con demostrar que un problema NP-Hard se puede reducir polinomialmente a C. Esto alcanza para demostrar que C también es NP-Hard, ya que si todos los problemas NP pueden ser reducidos polinomialmente a un problema Z que es NP-Hard y Z puede ser reducido polinomialmente a C, entonces por transitividad todos los problemas de NP pueden ser reducidos polinomalmente a C.

Por lo tanto, se harán ambas demostraciones para el problema Casting.

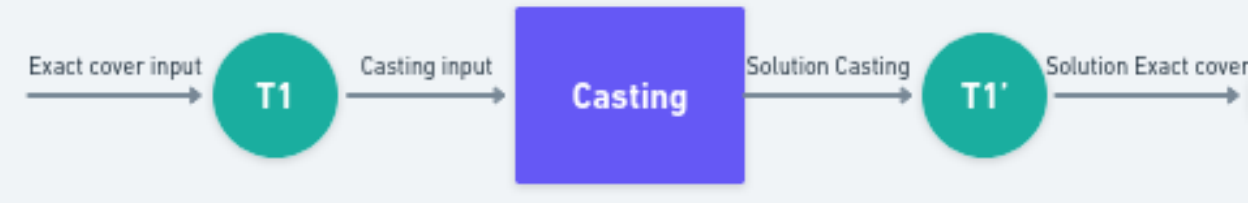
Demostración de que el problema Casting es NP

Para verificar una solución del problema Casting es necesario:

- Verificar que todos los participantes del conjunto encontrado como solución existan
- Al juntar todas las características que tengan los participantes seleccionados:
  - No debe haber características repetidas. Esto permite verificar que no haya participantes seleccionados que tengan las mismas características
  - Deben estar todas las características de la lista de las que se esperaba que tengan los participantes

La primera verificación se puede hacer recorriendo a los participantes de la solución verificando que cada uno esté en el problema original y las verificaciones de las características obtenidas al juntar las que tienen los participantes seleccionados se puede hacer buscando para cada característica que haya siempre un solo participante que la tenga. Dado que todo esto se puede hacer en tiempo polinomial, existe un algoritmo certificador polinomial para el problema Casting. Esto implica que el problema Casting es NP.

Demostración de que el problema Casting es NP-Hard



Para demostrar esto se encontrará una reducción polinomial del Exact cover al problema Casting, es decir se probará que $Exact\ Cover \leq_p Casting$. Dado que el problema Exact cover es NP-Hard, lo cual se demostrará en la sección siguiente, esta reducción alcanzá para demostrar que el problema del Casting es NP-Hard. Para hallar la reducción es necesario encontrar las transformaciones T1 y T1' tal que:

T1: Transformar la entrada del Exact cover en la del Casting

La entrada del Exact cover consiste en un conjunto X y una familia S de subconjuntos del mismo. La entrada del Casting consiste en un conjunto de características al que llamaremos C y una lista para cada una de ellas que contiene a todas las personas que poseen esa característica.

La primera parte de la transformación consiste en establecer que el conjunto X cumplirá el rol del conjunto de características C. De esta forma, los elementos del conjunto X pasaran a ser las características que se espera que tengan los participantes.

La segunda parte de la transformación consiste en decir que el rol de los participantes del Casting lo cumplirán los miembros de la familia S del problema original. Esto implica que para cada una de las características en C, que eran los elementos de X del problema original, será necesario encontrar el conjunto de participantes que la tiene, es decir una lista de miembros de la familia S. Esto ultimo no existe en el problema original, por lo que será necesario construirlo transformando las familias de subconjuntos en un conjunto para cada elemento de X que contenga las familias que en las que se encuentra.

Ejemplo:

A partir de la entrada del Exact cover:

$X = \{1, 2, 3\}$

$S = \{A1, A2, A3\}$ con:

- $A1 = \{1\}$
- $A2 = \{1, 2\}$
- $A3 = \{2, 3\}$

Primero, se establecerá que X cumplirá el rol de C, por lo tanto:

$C = \{1, 2, 3\}$

de este modo, las características que se espera que tengan los participantes son 1, 2 y 3.

Luego, se establece que los participantes del Casting serán los miembros de la familia S. Por lo tanto, los participantes del Casting serán A1, A2 y A3. Ahora será necesario construir para cada característica un conjunto de los participantes que la tiene. Esto implica ver para cada uno de los elementos de C, determinar en cuales de los conjuntos A1, A2 y A3 se encuentra:

- $1 = \{A1, A2\}$
- $2 = \{A2, A3\}$
- $3 = \{A3\}$

Esta transformación es polinomial ya que la única modificación que es necesario hacer en la entrada del Exact cover es la transformación de la familia de subconjuntos, la cual se puede hacer en tiempo polinomial recorriendo una vez X y S. Para cada elemento de X se creara un conjunto en el que se pondrán los elementos de S que lo contengan. Los otros cambios que implica la transformación son asignar los roles de cada cosa sin modificar los datos de la entrada en sí.

T1': Transformar la solución del Casting en la solución del Exact cover

La solución del problema del Casting es un conjunto de participantes. La solución del problema Exact cover es un subconjunto de miembros de la familia S. Dado que en T1 se estableció que el rol de los participantes lo cumpliran los miembros de la familia S, aquí simplemente se hará lo inverso: se dirá que el rol del conjunto de miembros de la familia S lo cumplirá el conjunto de participantes.

Estableciendo estas transformaciones se puede ver que la solución del Casting presenta las características que se esperan de la solución del Exact cover. Esto ocurre porque la solución del Casting es un conjunto de participantes que posee todas las características y en el que no existen 2 participantes con la misma característica. Dado que el conjunto de características cumple el rol del conjunto X y el conjunto de participantes el de los elementos de la familia S, esto implica que se obtuvo un conjunto de elementos de la familia S tal que la unión de sus conjuntos cubre a todos los elementos del conjunto X y son conjuntos disjuntos dado que no existen 2 conjuntos que tengan el mismo elemento de X. Esto ultimo es lo que se espera de la solución del Exact Cover, lo que implica que se lo redujo polimialmente al problema del Casting.

Ejemplo:

Continuando con el ejemplo de la sección anterior, los participantes elegidos para el Casting serán A1 y A2. Aquí simplemente lo que se hará establecer que A1 y A2 son los conjuntos de la familia S que se encontraron como solución del problema de Exact cover.

Esta transformación es polinomial ya que no requiere hacer ninguna modificación en la solución del Casting.

💬 Algunas referencias
https://www.geeksforgeeks.org/exact-cover-problem-algorithm-x-set-1/
https://en.wikipedia.org/wiki/Exact_cover
https://www.ntt-review.jp/archive/ntttechnical.php?contents=ntr201711fa3.html

2.

> 💡 Demuestre que EXACT-COVER es NP-C (puede ayudarse con diferentes problemas, entre ellos 3SAT, para hacerlo)

Demostración de que el Exact cover es NP

Para verificar una solución del Exact cover es necesario verificar que:

- Todos los elementos de la solución S’ pertenencen a la familia de conjuntos S
- Al juntar todos los elementos de los conjuntos que están en S’:
  - Se recuperan todos los elementos del conjunto X
  - No hay ningún elemento que no pertenezca a X
  - No hay ningún elemento repetido. Esto permite verificar que los conjuntos de S’ sean disjuntos.

Todas estas verificaciones se pueden hacer recorriendo los elementos de S’ y luego juntando los elementos en los conjuntos que estén allí, para luego compararlos con los de X. Dado que esto se puede hacer en tiempo polinonomial, existe un certificador polinomial para el Exact cover. Por lo tanto, el Exact cover es NP.

Demostración de que el Exact cover es NP-Hard

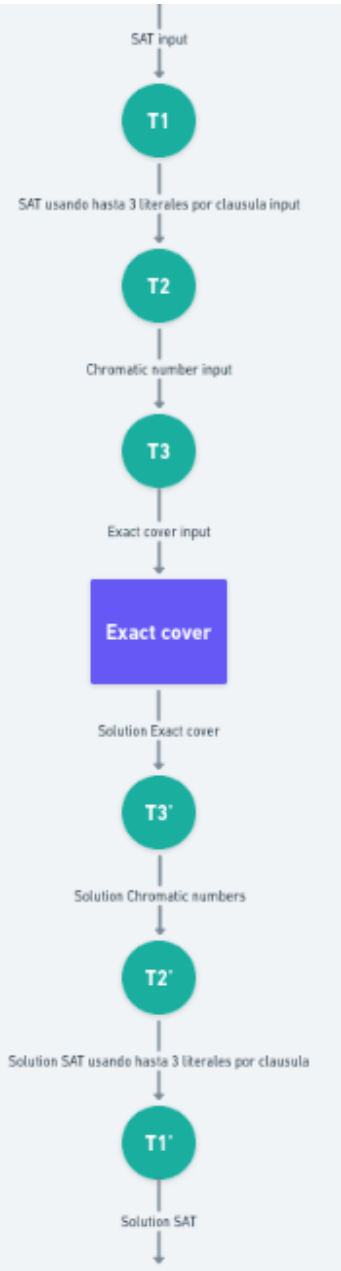
Los problemas que se utilizaran para la demostración son:

- **SAT (Boolean Satisfiability problem):** El problema SAT consiste en determinar si dada una expresión booleana existe una asignación de valores tales para que la expresión sea verdadera.
- **SAT usando hasta 3 literales por clausula:** Este problema es un caso especial del problema SAT. En esta versión, cada clausula de la expresión booleana tiene como máximo 3 literales.
- **Chromatic number:** Dado un grafo y un entero positivo *k*, consiste en determinar si es posible pintar todos los nodos del grafo utilizando *k* colores distintos para hacerlo, cumpliendo la regla de que no puede haber 2 nodos adyacentes pintados del mismo color.

Las versiones de los problemas que se consideran aquí son las versiones de decisión de los mismos. Esto quiere decir que para cualquier instancia del problema la respuesta ser SÍ o NO. Esto mismo ocurre con la versión que se considerará del Exact cover, que consiste en decidir si existe un cubrimiento exacto, a diferencia de la versión original del problema que consiste en encontrarlo. Sin embargo, todo lo que se demuestre es válido también para la versión original del problema, dado que la versión de decisión se puede reducir polinomialmente a esta. Esto es así ya que la solución del problema original se puede transformar en

la solución del problema de decisión: si existe una solución al problema original, entonces existe una cobertura mínima, mientra que si no existe la solución, no existe la costurera mínima tampoco.

Transformaciones



El Teorema Cook-Levin establece que si un problema $X \in NP$ entonces $X \leq_p SAT$. Es decir, el problema X puede reducirse polinomialmente a SAT. Si se consigue reducir polinomialmente el SAT a otro problema Y, por transitivitividad esto implicaría que todos los problemas de NP se pueden reducir

polinomialmente Y. Por esta razón, para demostrar que el problema Exact cover es NP-Hard se buscará reducir polinomialmente a este el problema SAT.

Para esto, se harán las reducciones

$$SAT \leq_p SAT\ usando\ hasta\ 3\ literales\ por\ clausula$$
$$\leq_p Chromatic\ number \leq_p Exact\ cover$$

lo que es equivalente a encontrar las transformaciones polinomiales:

T1: Transformar la entrada del SAT en la del SAT usando hasta 3 literales por clausula

Para que todas las clausulas tengan como máximo 3 literales, a las *a1 or a2 .. or am,* donde los literales son los *ai* y hay más de 3 literales se las reemplaza por:
(a1 or a2 or b1) (a3 or … or am or ~b1) (~a3 or b1) ... (~am or b1)

donde b1 es una nueva variable

Ejemplo:

La clausula *a1 or a2 or a3 or a4 or a5* se transformaría primero en:

(a1 or a2 or b1) (a3 or a4 or a5 or ~b1) (~a3 or b1) (~a4 or b1) (~a5 or b1)

luego la clausula (*a3 or a4 or a5 or ~b1)* que es la única con más de 3 términos se transformaría en:

(a3 or a4 or b2) (a5 or ~b1 or ~b2) (~a5 or b2) (b1 or b2)

Uniendo todo quedaría la clausula inicial transformada en:

(a1 or a2 or b1) (a3 or a4 or b2) (a5 or ~b1 or ~b2) (~a5 or b2) (b1 or b2)(~a3 or b1) (~a4 or b1) (~a5 or b1)

donde todas las clausulas tienen como máximo 3 literales.

Esta transformación se puede realizar en tiempo polinomial ya que la cantidad de pasos que será necesario hacer esta acotada por la cantidad de variables que hay multiplicada por la cantidad de clausulas, ya que habrá tantos pasos como literales “de más” haya en las clausulas, y el tiempo que se tarda en cada paso es proporcional a la cantidad de variables que tenga la clausula que se está transformando.

T1’: Transformar la solución del SAT usando hasta 3 literales por clausula en la del SAT

No es necesario hacer nada hacer ninguna transformación. Esto ocurre ya que el conjunto de las clausulas transformadas se puede satisfacer si y solo si se puede satisfacer el conjunto original de clausulas.

T2: Transformar la entrada del SAT usando hasta 3 literales por clausula en la del Chromatic number

1. Para cada una de las variable A*i* de las expresiones booleanas se crean los nodos *Vi, Vi’* y *Xi*

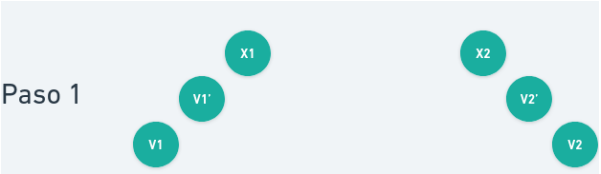
2. Para cada una de las clausulas *Cj* se crea el nodo *Cj* en el grafo
3. Para cada *i* se conecta *Vi* con *Vi'* con una arista
4. Para cada *i* y *j* se crea una arista entre *Xi* y *Xj*, cuando *i* sea distinto de *j*
5. Para cada *i* y *j* se crea una arista entre *Vi* y *Xj* y otra entre *Vi'* y *Xj*, cuando *i* sea distinto de *j*
6. Para cada *i* y *j* si *Aj* no aparece en *Ci* se crea una arista entre *Ci* y *Vj*
7. Para cada *i* y *j* si *~Aj* no aparece en *Ci* se crea una arista entre *Ci* y *Vj'*

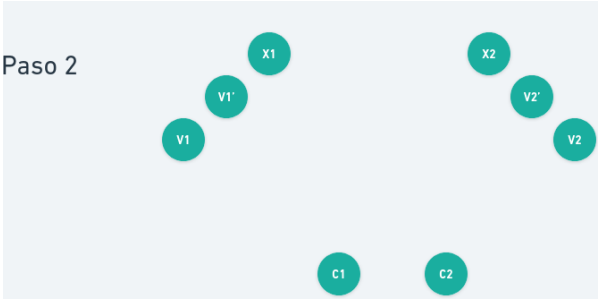
Ejemplo:

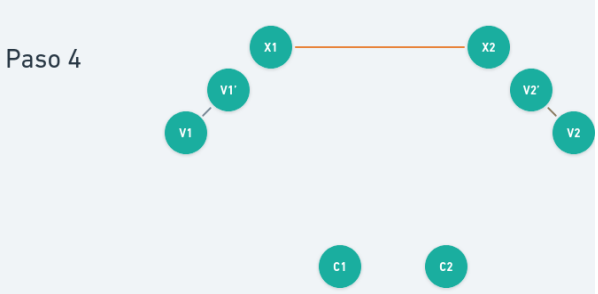
El armado paso a paso del grafo a partir de las clausulas:

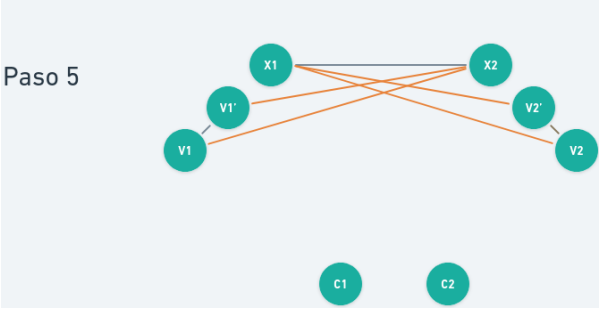
C1 = (A1 or A2) y C2=(~A1 or A2)

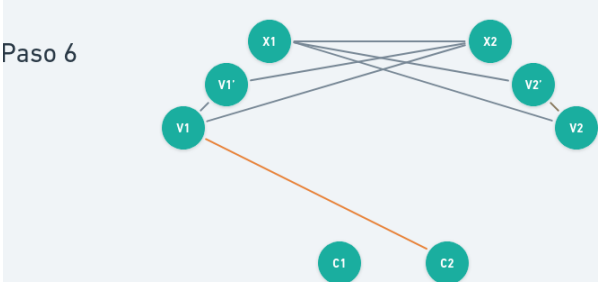
se puede hacer de la siguiente manera:

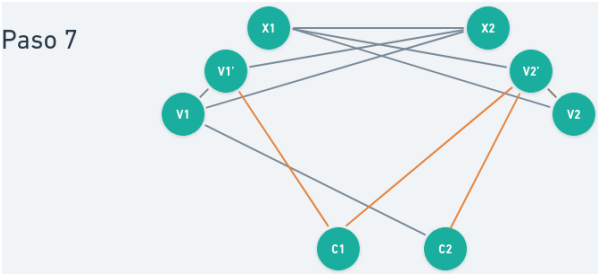












Esta transformación es polinomial ya que cada arista se puede crear en tiempo constante y la máxima cantidad de aristas que podría ser necesario incluir en el grafo es $\frac{n*(n-1)}{2}$ siendo n el triple de la cantidad de variables, dado que hay 3 nodos por cada una, sumado a la cantidad de clausuras

T2': Transformar la solución del Chromatic number en la del SAT usando hasta 3 literales por clausula

La formula booleana se puede satisfacer si se puede colorear el grafo construido usando *n+1* colores siendo *n* la cantidad variables en la formula booleana. Esta transformación se puede realizar en tiempo constante.

T3: Transformar la entrada del Chromatic number en la del Exact cover

A partir del grafo, se puede armar el conjunto X incluyendo en él:

- Todos los nodos
- Para cada arista del grafo, se incluye un elemento por cada numero entre 1 y *k*. Por ejemplo a partir de la arista 1 se agregan los elementos *E11*, *E12*, .. *E1K*

La colección de conjuntos S está formada por:

- Los conjuntos *Cij* con *i* entre 1 y la cantidad de nodos y *j* entre 1 y *k*. Las reglas para armar estos conjuntos son:
  - Los conjuntos *Cij* contienen al nodo *i* para todos los *j*
  - Si la arista *e* une los nodos *Va* con *Vb*, los conjuntos *Caj* y *Cbj* contienen *Eej* para cualquier *j* menor o igual a *k*

- Los conjuntos *Dij* que contienen elemento *Eij* para todo *i* y todo *j*

Ejemplo

Para el grafo:

Suponiendo que *k=2*, el conjunto *X* quedaría

$$X = \{V1, V2, V3, E_{11}, E_{12}, E_{21}, E_{22}\}$$

y la colección de conjuntos S sería

$S = \{C_{11}, C_{12}, C_{21}, C_{22}, C_{31}, C_{32}, D_{11}, D_{12}, D_{21}, D_{22}, D_{31}, D_{32}\}$ con:

$C_{11} = \{V1, E_{11}\}$

$C_{12} = \{V1, E_{12}\}$

$C_{21} = \{V2, E_{11}, E_{21}\}$

$C_{22} = \{V2, E_{12}, E_{22}\}$

$C_{31} = \{V3, E_{21}\}$

$C_{32} = \{V3, E_{22}\}$

$D_{11} = \{E_{11}\} \quad D_{12} = \{E_{12}\}$

$D_{21} = \{E_{21}\} \quad D_{22} = \{E_{22}\}$

$D_{31} = \{E_{31}\} \quad D_{32} = \{E_{32}\}$

Esta transformación se puede realizar recorriendo el grafo y agregando para cada nodo o arista lo que corresponda a los distintos conjuntos. Dado que ambas se pueden realizar en tiempo polinomial, la transformación es polinomial.

T3': Transformar la solución del Exact cover en la del Chromatic number

El problema de Chromatic Numbers se puede resolver usando *k* colores si el Exact cover tiene una solución. Esta transformación se puede realizar en tiempo constante.

💬 Algunas referencias

https://www.itu.dk/people/carsten/papers/chromatic.pdf

https://www.researchgate.net/profile/Michael-Haythorpe/publication/331397207_Linearly-growing_Reductions_of_Karp's_21_NP-complete_Problems/links/6162af9ce7993f536cb6b3b7/Linearly-growing-Reductions-of-Karps-21-NP-complete-Problems.pdf?origin=publication_detail

Introduction to the Design and Analysis of Algorithms: A Strategic Approach R. Lee, S. Tseng, R. Chang, Y. Tsai

3.

💡 Utilizando el concepto de transitividad y la definición de NP-C explique qué ocurriría si se demuestra que el problema EXACT-COVER pertenece a la clase P.

Si se demostrara que el problema de EXACT-COVER puede resolverse de forma polinomial (pertenece a la clase P), se demostraría que todos los problemas de NP pueden resolverse de forma polinomial. Esto último se debe a que *Exact Cover* es un problema NP-C y como se vio antes, por la definición de transitividad, todo problema NP puede ser reducido a un problema NP-C.

Si un problema $X \in NP$ entonces puede reducirse a un problema $X' \in NP_C$ y además $X' \in P$, por lo tanto $P = NP$.

Según Scott Aaronson en un artículo de su blog titulado `Reasons to believe`:

💬 *"If P=NP, then the world would be a profoundly different place than we usually assume it to be. There would be no special value in "creative leaps," no fundamental gap between solving a problem and recognizing the solution once it's found. Everyone who could appreciate a symphony would be Mozart; everyone who could follow a step-by-step argument would be Gauss; everyone who could recognize a good investment strategy would be Warren Buffett"*

4.

💡 Un tercer problema al que llamaremos X se puede reducir polinomialmente a EXACT-COVER, qué podemos decir acerca de su complejidad?

Poder reducir polinomialmente X al Exact Cover, indicaria que X es al sumo tan complejo como este. Si se encontrara una forma polinomial de resolver el Exact cover, entonces esto nos indicaría que hay una forma polinomial de resolver X.

5.

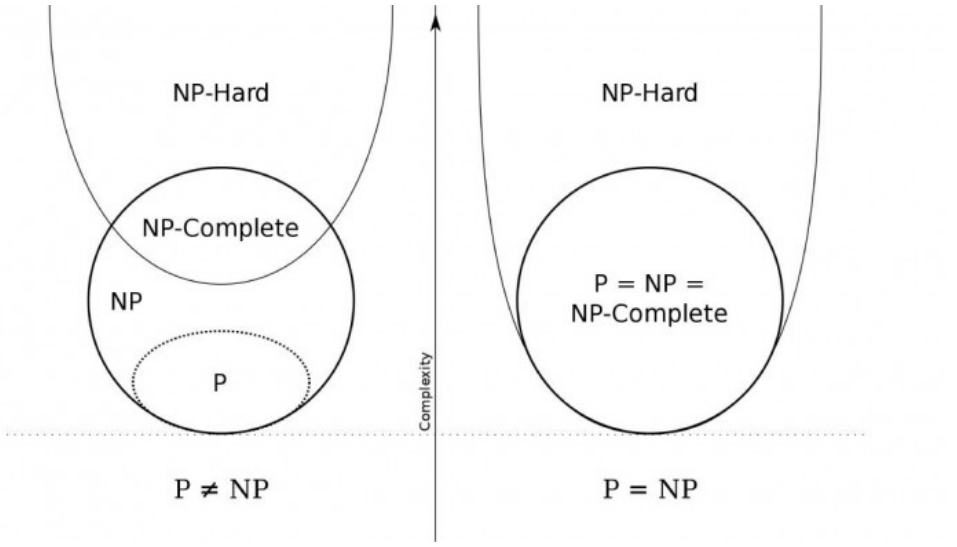
💡 Realice un análisis entre las clases de complejidad P, NP y NP-C y la relación entre ellos.

P: esta compuesto por todos los problemas que se pueden resolver en tiempo polinomial. Esto significa que existe una solución “eficiente” para ellos.

NP: esta compuesto por todos los problemas cuya solución se puede verificar de manera eficiente.

En problemas de decisión, se puede construir un certificador polinomial a partir de un algoritmo que resuelva el problema de manera polinomial. Por esta razón, todos los problemas que están en **P** estan tambien en **NP**. La inversa seria cierta solamente si **P=NP**.

NP-C: es la intersección entre los conjuntos de problemas **NP** y **NP-Hard**. Cuando un problema se encuentra en esta categoria, tiene sentido suponer que no tiene una solución polinomial, ya que esto implicaría que muchos problemas para los que hace años se trabaja sin exito en encontrar una solución polinomial tambien la tengan.



Correcciones primera entrega

Las correcciones hechas sobre la primera entrega son lo que está escrito en letra verde.

Nota Entrega:

Parte 1: Reentrega

Parte 2: 7,5

Corrector: Víctor

comentarios sobre las correcciones

Comentarios Parte 1

Presenta
en la misma pagina caratula, indice y enunciado de parte 1. Si lo hace por una cuestion de limitar a 20 paginas, tenga en cuenta que se indico que ni el indice, ni caratula cuentan dentro de esta limitación

Hace
una breve descripcion de ambos metodos. Ojo que cuando explica "Successive Shortest Path Algorithm" considera que hay varios nodos productores y varios consumidores. y ahi realiza una transformacion al problema de flujo maximo. en nuestro problema esto no se requiere puesto que ya es un problema de flujo máximo (y costo mínimo). Ojo con la afirmacion que solo se usa bellman ford para ver si hay ciclos negativos. inicialmente no hay ningun ciclo negativo. se puede usar dijsktra solo si se realiza una transformacion para evitar los ejes con peso negativos.

Selecciona finalmente "Cycle Cancelling Algorithm"

Afirma
"Seleccionaremos un nodo s como source y otro nodo t como sink ." Esto como impacta en nuestro problema particular donde tengo que transportar personas en vuelos de avion?

Brinda pseudocodigo y breve explicacion del método

Afirma "No se usan otras estructuras de datos" ademas del grafo. y para Bellman Ford?

Sobre
el analisis de complejidad temporal: "siendo C el costo total del flujo inicial". ¿Puede generalizar este valor de alguna forma? y no ser un valor del resultado de la ejecución?. Tiene que evaluar tambien el

tiempo de calcular el flujo maximo inicial. y ademas el costo de actualizar el flujo dentro del grafo residual en el camino de aumento en cada iteracion (puede o no afectar la complejidad final del algoritmo, pero para establecerlo se debe analizar). Puede afirmar que el algoritmo es polinomial o pseudopolinomial?

En el analisis de optimalidad:
A que se refieren con "el costo mínimo tiene una solución optima determinada"? y que "toda la data es integral" ? Que data? a que se refere con integral?. No es completo el analisis de optimalidad.

Por
que en el ejemplo seleccionar fuente y sumidero? puede ser cualquier nodo y el resultado no cambia? el ejemplo tendria que ser relacionado con el problema de los aviones.

En el ejemplo seria bueno agregar
cual es el costo total enviado, cual es el flujo total enviado. cual es el cuello de botella del ciclo encontrado. mostrar que esto no cambia el flujo máximo, pero que si disminuye el costo total.

Sobre el
analisis de la complejidad de lo programado: si programo BFS para encontrar el camino de aumento esta aplicando Edmonds-Karps y esto no tiene como complejidad el flujo maximo sino que termina siendo polinomial.

Programa Parte 1

El programa al ejecutarlo da error:

Traceback (most recent call last):

File "/home/vpode/tda1/2022-1c/tp3-e1/Grupo 45/main.py", line 1, in <module>

from grafo import *

File "/home/vpode/tda1/2022-1c/tp3-e1/Grupo 45/grafo.py", line 3, in <module>

from tkinter import E

ModuleNotFoundError: No module named 'tkinter'

Se elimina el modulo y se ejecuta correctamente

En todas las pruebas realizadas calcula bien el flujo máximo pero mal el costo mínimo.

Ejemplo:

A

G

A,B,1,10

A,C,2,10

A,D,3,10

B,E,1,3

C,E,2,3

C,F,2,3

D,F,3,3

E,G,0,5

F,G,0,5

Informa:

>> Costo Mínimo: 14, Capacidad Máxima: 10

(tendria que hablar de cantida de pasajeros y costo de los viajes)

En este caso podria enviar las siguientes personas

A->B->E->G : 3 personas con costo $2 por persona = $6

A->C->E->G : 2 personas con costo $4 por persona = $8

A->C->F->G : 3 personas con costo $4 por persona = $12

A->D->F->G : 2 personas con costo $6 por persona = $12

Total a pagar $38 para un total de 10 personas

Comentarios Parte 2

Explica correctamente el problema de Exact Cover. Brinda ejemplos del mismo

"El algoritmo de certificación de C puede verificar la solución de una instancia del

problema
en tiempo aceptable (polinomial)." --> Deberia poner "puede
verificar una posible solución de cualquier instancia del problema ..."

Detalla que pasos seguir para demostrar que el problema es NP-C

Presenta
un certificador polinomial del problema de castings. detalla los pasos a
seguir e indica que se puede hacer en tiempo polinomial (aunque no
indica cada operacion que complejidad tiene)

Presenta una
reduccion polinomial de Exact Cover a Casting. Muestra las dos

transformaciones requeridas en la misma. La misma es correcta y esta bien explicada y ejemplificada. Presenta referencias

Luego pasa a demostrar que Exact Cover es NP-C. presenta certificador polinomial (correcto). Para la demostracion de NP-H realiza un conjunto de reducciones en cadena: sat -> 3SAT -> numero cromatico -> Exact cover. No era necesario todas esas transformaciones.

En la transformacion de SAT a 3SAT no indica que pasa si una clausula queda con unicamente 2 variables (para 3SAT pedimos exactamente 3 variables diferentes por clausula

El ejemplo que da de transformacion de 3SAT a numero cromatico no es un 3sat, sino un 2sat. La transformacio es correcta. pero no termina explicando cual es la relacion de los colores resultantes con la posibilidad de satisfacer la expresion original. Tampoco explica como volver a la solucion del problema original

La reduccion de numero cromatico a Exact Cover es correcta. Sin embargo nuevamente no se detalla la relacion de los colores seleccionados con la solucion del problema inicial. Eso permite comprender como se termina interpretando la solucion

Punto 3: Bien explicacion y buena la cita

Punto 4: "Dado que no se conoce una forma polinomial de resolver el Exact cover, poder reducir polinomialmente X a este no nos dice nada acerca de la complejidad de X." --> Esto es incorrecto, nos dice que es a lo sumo tan complejo como EXACT COVER

Punto 5: Acotadas las explicaciones pero correctas. Se podria ayudar con una explicación visual