1 Enunciado del Proyecto

Democracia representativa en Pandora
En un pais llamado Pandora se tiene una democracia representativa en la que se forma un congreso con políticos que representan a los ciudadanos. Gracias a análisis de información de redes sociales, se puede predecir con qué políticos se sentiría representado cada ciudadano. Dado que cada congresista recibe un sueldo, usted ha sido contratado para determinar cual es el número mínimo de congresistas que deben hacer parte del congreso para que todos los ciudadanos se sientan representados.

Problema: Dados N ciudadanos, M políticos, y por cada ciudadano una lista de los políticos con los que el ciudadano se sentiría representado, determinar cual es la mínima cantidad de políticos necesarios para representar a todos los ciudadanos de Pandora y quienes podrían ser dichos políticos.

Ejemplo: Dado N=8, M=5 y la siguiente lista de políticos por cada ciudadano:
Ciudadano 1: 1, 3, 5
Ciudadano 2: 2, 5
Ciudadano 3: 1, 2, 4
Ciudadano 4: 2, 3, 4
Ciudadano 5: 1, 5
Ciudadano 6: 1, 4, 5
Ciudadano 7: 3, 4
Ciudadano 8: 3, 5
La respuesta sería de longitud 2 porque por ejemplo los políticos 4 y 5 representarían a todos los ciudadanos
OUTPUT: [4,5]

2 Especificaciones del proyecto

2.1 Entorno de Desarrollo y entorno de prueba.

Dada la necesidad de un desarrollo relativamente rápido, elegí Python para este proyecto, dado que es un

- **Sistema Operativo:** Ubuntu/Kubuntu 22.10 x86_64, Minimal Install. También probado con Fedora Linux 38 Workstation x86_64, por lo tanto se sabe que el programa en teoría podría correr en distribuciones basadas en Debian, Red Hat Linux o Arch Linux, no tengo un computador con Mac OS X o BSD para probar con otros sistemas Unix
- **Kernel:** Linux 5.19.0-38-generic y 6.2.0
- **Lenguaje de Programación:** Python 3.11.7 x86_64, Compilado desde apt, incluye python3-pip, También probado con la versión de Arch y DNF (el manejador de paquetes de Fedora/RHEL)
- **Librerías Utilizadas**
  - os
  - itertools
- **Procesador:** AMD Ryzen 5 3550H with Radeon Vega Mobile Gfx (8) @ 2.100GHz
- **IDE:** Visual Studio Code 1.76.2, instalado desde un paquete snapd. Incluye extensiones de Git y Python. En Manjaro fue instalado desde el AUR con el paquete visual-studio-code-bin

2.2 Estructura del Input y Output

A primera vista, este pareciese ser un problema bastante extraño, el input no pareciese seguir una estructura tan concreta como aquella de los primeros 2 proyectos, sin embargo, al extraer los valores 'N' y 'M' al inicio de cada caso, es mucho mas sencillo de leer. El ejemplo de este input proveido es:

```
3
8 5
1 3 5
2 5
1 2 4
2 3 4
1 5
1 4 5
3 4
3 5
5 5
1 2
4 5
3 4
2
3 5
3 3
1 2
2
2 3
```

como puede verse, no hay una estructura concreta para cada array. Esto se debe a que estos son arrays cuya longitud esta no predeterminada, sino establecida por la cantidad de políticos que representan a un ciudadano concreto: En otras palabras, esto esta representando no uno, sino dos grafos diferentes. Esto es un poco abstracto al inicio, pero tiene sentido al separarse en ciudadanos y políticos. Estos dos grafos estan interconectados por una lista de adyacencia, donde se guardan las preferencias de los ciudadanos hacia los politicos y los representados por cada politico. Estos dos grafos se interconectan uno con nodos del otro, y se referencian constantemente en la solución propuesta.

2.3 Lectura de datos

Según se nos fue informado por los monitores, la lectura se hará a partir de la lectura de archivos de texto y, por lo tanto, la estructura del archivo de Python esta pensada para poder leer varios archivos de texto secuencialmente. Para esto hemos importado la librería 'os' de las librerías base de Python. la estructura es bastante similar a aquella del primer proyecto. Sin embargo, la división de casos se hace de forma diferente, dado que a diferencia de dicho Proyecto, toma varios arrays dentro de un solo caso. Cada caso es conformado por una cantidad 'N' de Arrays. A partir de esta se puede

3 Diseño de la Solución

3.1 Conceptos Teóricos Importantes

3.1.1 NP Completitud

Un problema es NP-Completo si:

- es un problema de decision.
- se puede demostrar en tiempo polinomial cuando la respuesta es si.
- un algoritmo de fuerza bruta puede encontrar todas las soluciones.
- pcional: desciende o puede demostrarse a partir de la prueba formal de otro algoritmo

  Esta solucion cumple con todas estas opciones si se toma la decision de ëste set de candidatos representa todos los ciudadanos"como un problema de decisión y desciende de Vertex Cover, que es NP-C, por lo que se califica dentro de la categoría.

3.2 Conceptos Prácticos Importantes

3.2.1 Tipado dinámico

El lenguaje de Programación utilizado para resolver este algoritmo fue Python, el cuál tiene la capacidad de mantener una estructura de tipado dinámica, esta implica que podemos declarar una variable de un tipo y esta podrá ser interpretada como de otro tipo a la hora de compilar el código, otros lenguajes de Programación con esta carácteristica son Ruby y Javascript. Esto es importante para este problema porque estamos constantemente convirtiendo los tipos de los valores dados.

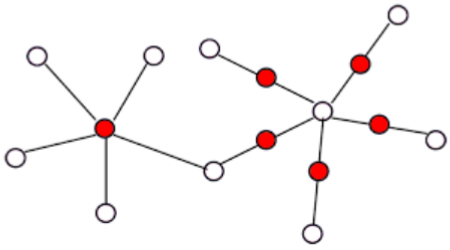
3.3 Problemas similares

3.3.1 Vertex Cover

Este problema desciende de Vertex cover, definido cómo [2]:

VERTEX COVER

Dado un grafo no dirigido G=(V,E)
el problema consiste en encontrar
V' como la **minima** cobertura de
vertices para G



donde se busca la cobertura mínima de vertices para acceder a todo el grafo. En este problema, sin embargo, se usa un grafo diferente para lograr acceder a la cobertura del grafo a evaluar, en lugar de usar los nodos de un solo grafo para resolver el problema.

Estrategia de solución Este problema por si mismo puede aproximarse de forma la cuál nos dé una solución de máximo el doble del tamaño de la solución óptima con el siguiente algoritmo:

APPROX-VERTEX-COVER(G)
1 $C = \emptyset$
2 $E' = G.E$
3 **while** $E' \neq \emptyset$
4 let (u, v) be an arbitrary edge of E'
5 $C = C \cup \{u, v\}$
6 remove from E' every edge incident on either u or v
7 **return** C

Dada la exactitud del resultado esperado, sin embargo, no se va a poder hacer esta aproximación en este caso, por lo cual aprovechando la NP-Completitud del caso, usaremos combinaciones y fuerza bruta para lograr solucionar este problema con un resultado óptimo, donde entre menor la longitud del array a ser respuesta, menor es el tiempo de ejecución.

3.4 La Solución

3.4.1 Mantras de diseño

- **El problema es NP-Completo:** Como ya fue explicado, este problema es una versión alternativa de vertex cover. Por lo que el problema va a ser NP-C, y la solución funcionará polinomialmente.
- **Es una solución especifica:** lastimosamente, no podemos utilizar una solución aproximada aunque Vertex Cover sea un algoritmo aproximado dada la especificación de que este algoritmo tiene que encontrar la solución óptima.
- **Hay dos grafos:** Para la solución propuesta, se convierte el input a dos grafos separados. dentro de un grafo, las conexiones no apuntan a otros nodos dentro del mismo sino a

3.4.2 Código Fuente

```
#Problema detectado: vertex cover
def solve(n, m, caso):

    politician_graph = {}
    citizen_graph = {}
    #generación primer grafo
    for i in range(n):
        citizen_graph[i+1] = caso[i]

    #generación segundo grafo.
    for i in range(1, m+1):
        citizens_into = []
        for j in range(1, n+1):
            check = citizen_graph[j]
            if i in check:
                citizens_into.append(j)
        politician_graph[i] = citizens_into

    i = 1
    while i < m:
        comb = combinations(politician_graph.keys(), i)
        for j in comb:
            voteset = []
            for k in j:
                voteset.extend(politician_graph[k])
            if len(set(voteset)) == n:
                return j
        i += 1
```

4 Análisis de la Solución

4.1 Complejidad

Nota: es posible que operaciones de entrada/salida y lectura/escritura fuera de la función principal en esta solución aumenten su tiempo de ejecución.
Temporal: Dado el uso de combinaciones para lograr esta solución, la complejidad temporal es $O(r * [nCr])$
Espacial: $O(n^2)$, por la creación de los grafos.

4.2 Casos Hipotéticos.

Suponga que conociendo el partido político al que pertenece cada político se quiere asegurar que haya al menos k partidos representados. A la hora de hacer las combinaciones, podemos filtrar el resultado a aquellos cuyo 'set' de partidos que se podría sacar por medio de la función 'filter' de python, cuyos especificos dependerían de la estructura del input y que sería de complejidad O(n). Se reemplaza el sistema de combinaciones filtrado y se opera con normalidad.

Suponga que se determina que cada ciudadano se siente representado por máximo dos políticos. Esta restricción técnicamente nos permitiría resolver el problema exactamente igual que como se resuelve acá, sin embargo, hay algunas optimizaciones que se abren a partir de esta restricción dado que con esta restriccion podemos intentar triangular el número de políticos que serán parte del sistema de soluciones anteriormente a computar las combinaciones. Esto reduciría significativamente el tiempo de ejecución de la aplicación pero no necesariamente su complejidad.

4.3 Limitantes

- el archivo solo puede recibir inputs de archivos de texto, un input no puede ser ingresado directamente desde el usuario a partir de la consola
- la necesidad de un Output exacto limita las capacidades de optimización de este algoritmo.

4.4 Consideraciones

- La capacidad de tipado dinámico de Python es directamente responsable de la capacidad de este algoritmo en algunos casos, gran parte de las funciones deberían reescribirse en caso de que se fuese a reescribir la solución en un sistema de tipado estático.
- Este programa asume un input correcto
- por ultima vez, este es un trabajo en grupo que realicé solo, por favor ten eso en cuenta, profe.

5 Referencias.

1 : *Enunciado del proyecto, ISIS 1105 Diseño y Análisis de Algoritmos Semestre 2023-10. Proyecto – PARTE 3* Uniandes, 2023.

2 *diaspositivas de clase.*