**Trabajo 2:**

Michael Moreno Valoyes

Alejandro Noriega Soto

Santiago Salazar Ramirez

**Punto 1:**

La organización de este punto se dividió en:

1.Entrada de datos.

1.1.Entrada del primer grupo.

1.2.Entrada del segundo grupo.

2.Explicación del algoritmo utilizado.

La entrada de datos empieza con pedir la cantidad de dependencias funcionales del primer grupo, luego de ello se introduce la parte izquierda y derecha de cada dependencia funcional.

Ejemplo:

Queremos introducir el grupo: [AP → BHW, BHW → C]

Para introducirlo: Izquierda 1 : A P (Se ve que va separado por espacios). Derecha1: B H W. Izquierda2: B H W. Derecha2: C

Lo mismo se hace en el grupo2.

Se utiliza el algoritmo presentado en las diapositivas de dependencias funcionales (Cierre de un atributo K bajo S: Ks+). Pero claro, ¿cómo se realizó? Tomamos el cierre como una lista, la cual empieza con el valor de la parte izquierda de la dependencia funcional. Luego de esto se revisa todas las dependencias funcionales del grupo a comprobar. En caso de que los atributos de la parte izquierda estén presentes en el cierre se toman los atributos de la parte derecha que no estén presentes en el cierre (se evitan duplicados claramente). El algoritmo proseguirá con su función (recorrer las dependencias funcionales del otro grupo de principio a fin) siempre y cuando el cierre se altere, en caso de que no se altere su tamaño se para el algoritmo (se verifica el tamaño del cierre al principio de cada recorrido completo y después de este). Ya cuando haya parado el algoritmo revisamos si la parte derecha de la dependencia funcional está presente en el cierre. En caso de que esté presente en el cierre significa que sí funciona según el otro grupo, en su contrario pues no.

Ya en nuestro caso específico lo que hacemos con el algoritmo es iterar el S1 (grupo 1) con todo el S2 (grupo 2). ¿Cómo así? Revisa cada dependencia funcional del grupo 1 con todas las dependencias funcionales, para luego verificar cada dependencia funcional del S2 con todo el S1. En caso de que en los dos grupo todas sus dependencias funcionales sí funcionasen con su grupo opuesto significa que estos dos grupos de dependencias funcionales son equivalentes, en caso de que esto no es así significa que no son equivalentes.

**Punto 2A:**

Se empieza con la entrada de datos, primero se escribe la cabecera de la relación a verificar, separados por un espacio.

Ejemplo: Digite la cabecera por favor

Cedula Nombre

Luego se pedirá por columnas lo datos de estas (asegúrese de que tengan el misma tamaño las columnas) también separados por un espacio

Ejemplo:

Ahora introduzca los valores por cada atributo:

Atributo Cedula:

12 23 76

Atributo Nombre:

Juan Juan Lisa

En ese momento se mostrará la tabla generada con la librería tabulate y se mostrará el resultado. Cuando se recibe el input de los datos de la cabecera se guardan en una lista, y con esta se forma un diccionario en el que cada key es un atributo y su valor es su columna, recorro las keys y para formar todos los posibles atributos compuestos hago el conjunto potencia de la lista de atributos y reviso si estos son claves candidatas con una función que:

- Si el atributo no es compuesto solo reviso si cada dato en su columna no está más de una vez en la columna, si no sucede esto, lo agrego a la lista de claves candidatas

- Si el atributo es compuesto primero reviso su irreducibilidad con el conjunto potencia del atributo y con estos reviso si no están ya en la lista de claves candidatas, si es irreducible realizo el mismo proceso formando tuplas de valores y revisando si estos no están en la lista de tuplas de valores que voy formando, si las tuplas no se repiten entonce va a la lista de claves candidatas

Ya con la lista de claves candidatas formo la string que las indica en el resultado con un ciclo for y procedo a hacer la lista de determinantes, para esta lista utilizo la función y lógica del punto2B y reviso con cada combinación de atributos si son o no dependencias funcionales completas, si lo son van a la lista de determinantes para iterar sobre esta y revisar si todos sus integrantes son claves candidatas (están en la lista de claves candidatas) entonces es BCNF y se muestra el resultado adecuado, de lo contrario se muestra que no es BCNF pero también se muestran las strings formadas con las claves candidatas y los determinantes.

**Punto 2B:**

La organización para este punto se dividió en:

1. Entrada de datos.

1.1.Entrada de datos de la tabla

1.2.Visualización de la tabla.

1.3.Entrada de datos de la verificación de la dependencia funcional.

2.Verificación de si es dependencia funcional.

3.Verificación de si es dependencia funcional completa.

La entrada de datos empieza con pedir lo que es la tabla, empieza primero con la cabecera, para seguir con los valores de cada atributo. ¿Cómo se introducen estos valores? Se introducen dando un espacio entre cada uno, esto para poder dividir los valores en una lista de una manera sencilla.

Ejemplo:

Cabecera: Nombre Cédula Dinero

Nombre: José Mateo Jorge

Cédula: 111 222 333

Dinero: 200 200 300

Por términos meramente de belleza se introdujo una librería encargada de dibujar la tabla que se acabó de introducir, esto para permitir al usuario (profesor) la sencilla visualización de los datos.

Por eso mismo es que se le recomienda al profesor que descargue la librería tabulate, la cual se descarga usando el siguiente comando en la consola (cmd): pip install tabulate. Igualmente ya se introdujo dicha librería en la entrega, en caso de que no desee hacer esto.

Luego de los datos de la tabla ya viene el introducir la dependencia funcional que se quiere verificar (el programa le preguntará si quiere hacerlo o no), empezando con la petición de la parte de la izquierda, para luego seguir con la parte de la derecha.

Ejemplo:

Usted quiere verificar (Nombre → Cédula), entonces lo que usted hace es:

Dependencia izquierda: Nombre

Dependencia derecha: Cédula

En caso de que quiera más de un elemento en alguna de sus partes lo único que se hace es dar un espacio entre ellos.

Segundo ejemplo:

Quiere verificar: (Nombre,Cédula → Dinero), entonces realiza:

Dependencia izquierda: Nombre Cédula

Dependencia derecha: Dinero

Ya la verificación de si es o no es dependencia funcional lo hago de una manera sencilla: verifico cada valor que tenga los atributos de la dependencia izquierda con la parte derecha. ¿Cómo? Con el uso de listas. ¿Cómo se hace de forma más explicita? La lista depende del conjunto de valores en donde está el recorrido (dado que empieza según el orden de la tabla). En caso de que la parte de la izquierda se encuentre en esa lista se verifica la parte de la derecha actual con la parte derecha (en donde andamos ubicados en la tabla) de la lista, en caso de que sea diferente significa que no es dependencia funcional. En caso de que la parte izquierda no esté en la lista se agrega la parte izquierda y derecha que estaban en el recorrido actual (puesto no hay para comparar, significa que es el primer conjunto de datos).

Para verificar si es completo se hace verificación de dependencia funcional con todos los subconjuntos de la parte izquierda. En caso de que existe uno que es dependencia funcional pues significa que no es dependencia funcional completa.

**Notas:**

1.El programa se ejecuta en Entregable.py, el cual tiene toda la interfaz necesaria para ejecutar los programas (los demás .py son las soluciones de los puntos, pero sin la interfaz). De otra forma más sencilla: el Entregable.py es el main.

2.Para los valores debe escribirlo como NONE y SIEMPRE debe de ser así, no puede cambiar la notación para este en ningún punto.

3.Le entregamos el tabulate.py para que no le toque descargar el tabulate.py con el pip install, en caso de que lo quiera instalar así no existe ningún inconveniente.

4.Como se hace el split() según el espacio se OBLIGA al usuario que si quiere un nombre con más de una palabra separada deba usar otra forma de escribir esto. Puede ser con un “\_”,”-”,”documentoDeIdentidad”,etc. Haga la opción que quiera, pero el nombre NO debe de estar separado por espacios si viene siendo del mismo nombre.