Taller 2: DFS, BFS y AGM

Fecha del taller: 19 de mayo de 2023

Este taller consta de 4 ejercicios tomados de UVa online judge. Estos pueden resolverse empleando las herramientas y conceptos que vimos relacionadas a DFS, BFS y AGM. Los objetivos puntuales del taller son:

- Implementar las distintas ideas que fueron desarrolladas durante las prácticas.
- Validar la correctitud y performance de las soluciones propuestas mediante un juez en instancias reales, y no solo teóricamente.
- Despejar cualquier duda que surja con los docentes.
- Seguir practicando las distintas técnicas aprendidas.

Si bien durante el taller el juez es el método de validación de los algoritmos propuestos, recomendamos fuertemente intentar comprender y justificar la correctitud del código desde un punto de vista más formal. Específicamente, se recomienda:

☐ Diseñar y pensar los algoritmos antes de empezar a programar.

☐ Estimar la complejidad computacional de los algoritmos propuestos.

☐ Esbozar un argumento claro que justifique la correctitud del algoritmo o del modelado.

Para cada uno de los 4 problemas se provee una traducción y resumen del enunciado, asi como los links a UVa y vjudge (cuando este segundo esté disponible), desde donde se pueden submittear las soluciones[1]. **Ambos jueces soportan hasta C++11**.

También se provee un "Ejercicio 0" en forma de ejercicio guiado para implementar el algoritmo de Kosaraju, el cual puede resultar altamente útil para el TP2.

[1] Damos ambos links para protegernos del caso en que la página UVa esté caída

Ejercicio 0: *Trust groups*[2][3]

Dado un conjunto V de empleados se los quiere subdividir en oficinas de tal forma de que en cada oficina solo haya empleados que *confían entre sí*.

Nos informan una lista de pares E: cada par (v, w) denota que v confía en w. Notemos que v puede confiar en w, y w **no confiar** en v (i.e. la relación no es simétrica). Para no tener que darnos todos los *pares de confianza* nos indican que si (v, w) y (w, z) son pares de confianza, entonces v confía en z, aunque (v, z) no sea un par dentro de E.

Dados V y E tenemos que calcular la menor cantidad de oficinas que se necesitan para dividir a los empleados en grupos *estables*, en donde cada par de empleados confía entre sí.

Descripción de una instancia

La entrada consta de múltiples casos de test. Cada caso de test comienza con un par N y M con $1 \leq N \leq 1000$ y $1 \leq M \leq 999000$. La entrada termina con una línea de la forma `0 0`.

Cada caso de test empieza con N líneas indicando los nombres y apellidos de cada una de las personas de la empresa. Luego siguen $2M$ líneas: cada par de líneas consecutivas denota un *par de confianza*. Es decir, si en la línea 1 está el nombre v y en la 2 la línea w, entonces (v, w) es un par de confianza. De la misma forma, las líneas 3 y 4 indican el siguiente par de confianza.

Descripción de la salida

Para cada caso de test hay que imprimir un único entero indicando la mínima cantidad de oficinas que se necesitan para organizar a los empleados en grupos estables.

Ejemplo de entrada y salida Se presenta un ejemplo de entrada y su correspondiente salida.

[2] UVa: https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=2756, vjudge: https://vjudge.net/problem/UVA-11709

[3] Este ejercicio es guiado, recomendamos leer su explicación y limitarse a implementar el algoritmo descripto.

Entrada de ejemplo	Salida esperada de ejemplo
3 2 McBride, John Smith, Peter Brown, Anna Brown, Anna Smith, Peter Smith, Peter Brown, Anna 3 2 McBride, John Smith, Peter Brown, Anna Brown, Anna Smith, Peter McBride, John Smith, Peter 0 0	2 3

Solución Con los nombres de las variables el problema nos sugiere armar un digrafo $D = (V, E)$ donde V es exactamente el conjunto de empleados y E la lista de *pares de confianza*. Podemos decir que dos empleados z y u confían entre sí si hay un camino de z a u y otro de u a z en D. Por lo tanto, nos podemos olvidar de las oficinas y los empleados, y simplemente pensar el problema de particionar los nodos de V en la mínima cantidad de partes $P_1, \ldots, P_k$ de tal forma que para todo par de nodos v, w en P_i valga que v tiene un camino hacia w y w uno hacia v, para $1 \leq i \leq k$.

Como primera observación, notemos que la relación $\mathcal{R}(v, w) \equiv$ *"hay un camino de v a w"* es una relación de equivalencia, y por lo tanto particiona a los nodos[4]. Más aún, estas componentes de nodos que se alcanzan entre sí se conocen como **Componentes Fuertemente Conexas (CFC)**. Un digrafo que puede ser muy útil en problemas que involucran estas componentes es el **digrafo de CFCs**: es un nuevo digrafo $D_{CFC} = (V_{CFC}, E_{CFC})$ donde V_{CFC} es el conjunto de componentes fuertemente conexas de D y dadas dos componentes $C_1, C_2 \in V_{CFC}$ vale que $C_1C_2 \in E_{CFC}$ si hay un nodo v en C_1 y otro w en C_2 de tal forma que vw sea un eje de D. Para un ejemplo, ver la figura 1.

[4]Lo deberían recordar de Álgebra I, pero, en todo caso, alcanza con saber que hay una única forma de armar las particiones $P_1, \ldots, P_k$ de tal forma que cada una sea *maximal*.

Figura 1: Llamemos D al digrafo de la izquierda. El conjunto $\{1, 2, 3, 4\}$ es una componente fuertemente conexa: dado cualquier par de nodos $1 \leq v, w \leq 4$ vale que hay un camino de v a w. El digrafo tiene dos componentes fuertemente conexas más: los conjuntos $\{5\}$ y $\{6, 7\}$. Observen que algunas CFCs no son ciclos (la componente {1,2,3,4}, por ejemplo). En la derecha tenemos el digrafo D_{CFC}, el cual se obtiene intuitivamente "colapsando" las componentes fuertemente conexas de D a un único nodo. Hay un eje entre las componentes 5 y 67 debido a que el nodo 5 tenía un eje hacia 6 en D.

El **algoritmo de Kosaraju** es capaz de encontrar las componentes fuertemente conexas de un dado digrafo en tiempo lineal. El mismo se basa conceptualmente en el algoritmo para encontrar un orden topológico que vimos en clase basado en DFS y el uso de un stack. Este consiste en:

1. Ejecutar DFS sobre D encolando en un stack S los nodos al momento en que se pintan de negro.
2. Ejecutar DFS sobre D^T[5] en el orden en que se van *poppeando* los nodos de S. Cuando se ejecuta DFS sobre un nodo no descubierto v todos los nodos pintados en esa ejecución forman una nueva componente fuertemente conexa.

La correctitud de este algoritmo la pueden leer del apunte de la clase 9,5 del campus. En esas diapositivas se demuestra en particular que D_{CFC} es siempre acíclico.

Sigamos la ejecución del algoritmo en el digrafo de la figura 1:

1. Primero ejecutamos DFS sobre sobre D, tomando un orden cualquiera de los nodos. Si primero procesamos el nodo 3, el stack S[6] nos queda como $S = \{4, 1, 2, 3\}$ (el nodo 3 es el último *pusheado*, ya que es el primero que se empezó a procesar). Si luego el DFS se ejecuta sobre 5 entonces se descubre todo lo que falta del digrafo, y el stack queda como $S = \{4, 1, 2, 3, 7, 6, 5\}$.
2. Ahora tenemos que correr DFS sobre D^T en el orden en que vamos *poppeando* los nodos de S. Primero lo ejecutamos sobre 5. Este DFS descubre un único nodo, el 5, y por lo tanto marcamos a $\{5\}$ como una CFC. El siguiente DFS se ejecuta sobre 6, y se descubren a los nodos 6 y 7 (5 ya estaba descubierto), encontrando la componente $\{6, 7\}$. El siguiente DFS sobre 7 no hace nada ya que 7 había sido descubierto. Finalmente, el DFS sobre 3 descubre a los nodos 1, 2, 3 y 4, que forman la última CFC que buscábamos. El resto de los DFSs no hacen nada.

Ayudados por el algoritmo de Kosaraju tenemos que terminar de programar el algoritmo del archivo `trust_groups.cpp`. En el mismo ya está escrita la tediosa parte de armar el digrafo D y D^T a partir de los nombres de las personas y los *pares de confianza*.

[5] D^T es el digrafo que se obtiene "girando" los ejes de D.
[6] Escribo el stack de izquierda a derecha, i.e. el *top* del stack es el último elemento a la derecha.

Ejercicio 1: *Oil Deposits*[7]

En un mapa con forma de grilla tenemos indicada la posición de varios pozos de petróleo. Para organizar el mapa en regiones se define el siguiente criterio: dos pozos pertenecen a una misma región si son adyacentes horizontal, vertical o diagonalmente.

Dado el mapa, nuestro objetivo es contar la cantidad de regiones que se van a armar con este criterio.

Descripción de una instancia

La entrada consiste de muchos casos de test.

Cada uno comienza con una línea con dos números n y m, con $1 \leq n, m \leq 100$.

Luego sigue la descripción del mapa de dimensión $n \times m$ correspondiente al caso de test. Las posiciones con pozos se indican con el caracter @, mientras que las posiciones vacías se describen con el caracter $*$.

La entrada termina con una línea de la forma `0 0`.

Descripción de la salida

Para cada caso de test se debe imprimir una línea indicando la cantidad de regiones que se crearán si estas se definen con el criterio nombrado anteriormente.

Ejemplo de entrada y salida Se presenta un ejemplo de entrada y su correspondiente salida.

Entrada de ejemplo	Salida esperada de ejemplo
1 1 * 3 5 *@*@* **@** *@*@* 1 8 @@****@* 5 5 ****@ *@@*@ *@**@ @@@*@ @@**@ 0 0	0 1 2 2

[7] Uva: https://onlinejudge.org/index.php?option=onlinejudge&Itemid=8&page=show_problem&problem=513, vjudge: https://vjudge.net/problem/UVA-572

Ejercicio 2: *Unlock the Lock*[8]

Tuki encontró un candado de 4 dígitos en la parada del 42 y, naturalmente, se dispuso a encontrar la clave para abrirlo. Para su sorpresa, un número de 4 dígitos había sido escrito detrás del mismo, así que sospecha que ese es el código y que lx dueñx no es una persona muy astuta.

Sin embargo, la combinación del candado no se puede mover arbitrariamente. Hay un conjunto de diminutos botones con números impresos sobre ellos que cambian el valor de la combinación: si se presiona el botón con el número x entonces se le suma x a la clave actual, dando el apropiado giro si al sumar x se excede la representación numérica del candado (es decir, si a la combinación 9999 se le suma 1 luego se obtiene 0000).

Tenemos que diseñar un programa para ayudar a Tuki a encontrar la menor cantidad de botones que debe tocar para alcanzar la (supuesta) clave del candado.

Descripción de una instancia

La entrada contiene muchos casos de test.

Cada uno comienza con una línea con 2 combinaciones de 4 dígitos L y U y un número R con $1 \leq R \leq 10$. L es la combinación actual en la que se encuentra el candado, y U la escrita por detrás del mismo. R denota la cantidad de botones que tiene el candado.

En la siguiente línea hay R números, cada uno en correspondencia con uno de los botones del candado. Cada uno de estos números r_i cumple que $0 \leq r_i \leq 9999$.

La entrada termina con una línea de la forma `0 0 0`.

Descripción de la salida

Para el caso de test i se debe imprimir la línea `Case` i`:` k donde k es la mínima cantidad de botones que debe presionar Tuki para llevar la combinación L a U. En caso de que no sea posible, k debe ser el mensaje *Permanently Locked*.

Ejemplo de entrada y salida Se presenta un ejemplo de entrada y su correspondiente salida.

Entrada de ejemplo	Salida esperada de ejemplo
0000 9999 1 1000 0000 9999 1 0001 5234 1212 3 1023 0101 0001 0 0 0	Case 1: Permanently Locked Case 2: 9999 Case 3: 48

[8] UVa: https://onlinejudge.org/index.php?option=onlinejudge&Itemid=8&page=show_problem&category=0&problem=3312&mosmsg=Submission+received+with+ID+28476442, vjudge: https://vjudge.net/problem/UVA-12160

Ejercicio 3: *Racing*[9]

Debido al inminente y furioso estreno de *Fast X* el gobierno de la ciudad se quiere preparar para un incremento masivo de las carreras callejeras. Se tomó la decisión de colocar cámaras en distintas calles, de forma tal de desincentivar la participación en dichos eventos.

Naturalmente, el gobierno está interesado en gastar la menor cantidad de dinero posible. Un equipo de expertos ya recorrió la ciudad y nos informó, para cada calle, el costo de poner una cámara en ella. En base a un exhaustivo análisis de las últimas nueve películas este mismo equipo concluyó que todas las carreras peligrosas ocurren sobre circuitos, es decir, comienzan y terminan en el mismo lugar.

Con toda esta información el gobierno nos encomienda la siguiente tarea: calcular la mínima cantidad de dinero que se requiere para colocar suficientes cámaras de tal forma que no quede ningún circuito en la ciudad sin vigilancia.

Es importante aclarar que los corredores no respetan el sentido de las calles, y por lo tanto el modelo de la ciudad no está orientado. Aparte, se sabe que las carreras callejeras no repiten calles, ya que esto podría provocar choques entre los primeros y los rezagados.

Descripción de una instancia

La entrada tiene muchos casos de test. La primera línea contiene un entero c indicando su cantidad.

Cada caso de test comienza con una línea con dos enteros $1 \leq n \leq 10000$ y $1 \leq m \leq 100000$, indicando la cantidad de esquinas de la ciudad y la cantidad de calles. Luego siguen m líneas, cada una con 3 enteros v, w y c, denotando la existencia de una calle entre v y w de costo c. Las esquinas serán representadas con números entre 1 y n, y el valor de cada costo e un entero no negativo acotado por 1000.

Al final de la entrada hay un `0`[10].

Descripción de la salida

Por cada caso de test se debe imprimir un entero indicando el mínimo costo necesario para colocar cámaras de tal forma que no quede circuito sin cubrir.

Ejemplo de entrada y salida Se presenta un ejemplo de entrada y su correspondiente salida.

[9] Uva :https://onlinejudge.org/index.php?option=onlinejudge&Itemid=8&page=show_problem&problem=3675, vjudge: https://vjudge.net/problem/UVA-1234

[10] Yo tampoco entiendo por qué.

Entrada de ejemplo	Salida esperada de ejemplo
1 6 7 1 2 5 2 3 3 1 4 5 4 5 4 5 6 4 6 3 3 5 2 3 0	6

Ejercicio 4: *Hedge Mazes*[11][12]

La Reina de Nlogonia es fanática de los laberintos, y por lo tanto sus arquitectos construyen varios de estos alrededor de su castillo. Cada laberinto construido para la reina está compuesto de habitaciones conectadas por pasillos. Cada pasillo conecta un par de habitaciones, y puede ser recorrido en ambos sentidos.

Los súbditos de la reina le preparan un desafío distinto cada día, el cual consiste en encontrar, dada una habitación de inicio y una de final, un camino simple que las una. Un camino simple es una secuencia de habitaciones distintas tales que cada par de habitaciones consecutivas tienen un pasillo que las une. Naturalmente, la primera habitación del camino debe ser la habitación de inicio, mientras que la última debe ser la habitación final. La Reina considera que un desafío es *bueno* cuando, considerando todos los posibles recorridos que comienzan en la habitación inicial y terminan en la habitación final, exactamente uno de ellos es un camino simple.

Para ayudar a los súbditos de la Reina escriba un programa que dada la descripción de un laberinto y una serie de consultas especificando la habitación inicial y final determine, para cada consulta, si esa elección de habitaciones conforma un desafío *bueno*.

Descripción de una instancia

La entrada posee múltiples casos de test. Cada uno comienza con una línea que contiene 3 enteros R, C y Q, con $2 \leq R \leq 10^4$, $1 \leq C \leq 10^5$ y $1 \leq Q \leq 1000$. Cada uno representa, respectivamente, la cantidad de habitaciones, de pasillos, y de consultas. Las habitaciones se identifican con números naturales distintos entre 1 y R. Cada una de las siguientes C líneas describe un pasillo usando dos enteros distintos A y B, indicando que hay un pasillo que conecta A con B, con $1 \leq A < B \leq R$. Luego, cada una de las siguientes Q líneas describe una consulta usando dos enteros distintos S y T, indicando respectivamente la habitación inicial y final, con $1 \leq S < T \leq R$. Se puede asumir que para cada caso de test hay a lo sumo un pasillo entre cada par de habitaciones, y que no hay ninguna consulta repetida.

El último caso de test está seguido por una línea con 3 ceros.

[11] Uva :https://onlinejudge.org/index.php?option=onlinejudge&page=show_problem&problem=3785, vjudge: https://vjudge.net/problem/UVA-12363

[12] Tomado del TP2 de 2c2022.

Descripción de una salida

Para cada de test se deben imprimir $Q+1$ líneas, donde en la i-ésima línea se debe escribir la respuesta a la i-ésima consulta. Si las habitaciones inicial y final representan un desafío *bueno*, entonces se debe escribir el caracter 'Y'. En caso contrario, se debe escribir el caracter 'N'. Al final de cada caso de test se debe escribir un guión '-'.

Entrada de ejemplo	Salida esperada de ejemplo
6 5 3 1 2 2 3 2 4 2 5 4 5 1 3 1 5 2 6 4 2 3 1 2 2 3 1 4 1 3 1 2 0 0 0	Y N N - N Y Y -