Trabajo Práctico 2: Algoritmos sobre grafos

Compilado: 2 de noviembre de 2022

Fecha de entrega: 11 de noviembre de 2022 (antes de las 23:59)

Fecha de reentrega: 28 de noviembre de 2022 (antes de las 23:59)

Este trabajo práctico consta de cuatro ejercicios. En tres de estos cuatro ejercicios, parte de la evaluación consiste en el enviar el código escrito a un juez online y pasar una batería de tests en el tiempo estipulado. Para la aprobación del trabajo práctico se debe entregar:

1. Un informe *de a lo sumo diez páginas* que describa la solución de los ejercicios y responda a todas las consignas del enunciado.

2. El código fuente con los programas que implementan las soluciones propuestas a los ejercicios.

3. El link a su página de usuario en https://onlinejudge.org, donde se vean las entregas aceptadas para los problemas 1, 2 y 3[1].

El informe debe ser autocontenido, lo que significa que cualquier estudiante potencial de AED3 que conozca los temas debe poder leerlo sin necesidad de conocer cuál fue el enunciado. En particular, todos los conceptos y notaciones utilizados que no sean comunes a la materia deben definirse, incluso aquellos que se encuentren en el presente enunciado. No es necesario explicar aquellos conceptos propios de la materia ni la teoría detrás de las distintas técnicas algorítmicas o de demostración de propiedades.

El informe debe estar estructurado usando una sección independiente por cada ejercicio resuelto. Esto no impide que un ejercicio haga referencias a la solución de otro ejercicio; las comparaciones son muchas veces bienvenidas o parte de la consigna. Cada sección debe dar una respuesta cabal a todas las preguntas del enunciado. Sin embargo, el informe no es una sucesión de respuestas a las distintas consignas, sino una elaboración única y coherente donde se pueden encontrar las respuestas a las preguntas. La idea es que el informe sea de agradable lectura.

El informe **no debe incluir código fuente**, ya que quienes evaluamos tenemos acceso al código fuente del programa. En caso de utilizar *pseudocódigo*, el mismo debe ser de alto nivel, evitando construcciones innecesariamente complicadas para problemas simples.[2] Todo pseudocódigo debe estar acompañado de un texto coloquial que explique lo que el pseudocódigo hace en términos conceptuales.

El código fuente entregado debe venir acompañado de un documento que explique cómo compilar y ejecutar el programa. El código fuente debe compilar y debe resolver correctamente **todos** los casos de test que se le presenten en un tiempo que se corresponda a la complejidad esperada. En todos los ejercicios, la instancia se leerá de la entrada estándar y la solución se imprimirá en la salida estandar. Junto a este enunciado se incluyen algunos casos de test, mientras que la complejidad temporal en peor caso, de ser solicitada, forma parte del enunciado. Tener en cuenta que el programa puede ser probado en casos de test adicionales.

Para aprobar el trabajo práctico es necesario aprobar cada ejercicio en forma individual, ya sea en la primera entrega o en el recuperatorio. No es necesario reentregar aquellos ejercicios que sean

[1] Para generar esta página, primero se debe determinar el id de usuario propio, ese dato está en la sección de "My account" bajo el nombre *Online Judge ID*. Luego, se puede acceder por medio del link https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_authorstats&userid=<id>, reemplazando el "<id>" con el identificador obtenido.

[2] E.g., "tomar el máximo del vector V" vs. "poner max := V_0; para cada $i = 1, \ldots, n-1$: poner max := maximo(max, V_i);

aprobados en la primer entrega. Asimismo, para aprobar cada ejercicio es necesario que el informe describa correctamente la solución y que el programa propuesto sea correcto. La correcta escritura del informe forma parte de la evaluación.

El código entregado debe ser original, salvo por aquellos fragmentos tomados de códigos vistos en clase durante este cuatrimestre. Caso contrario, el TP será automáticamente desaprobado. Aparte, el acto de plagio puede llevar a consecuencias más graves a nivel institucional, como la expulsión de la materia.

Formato de entrega Para realizar la entrega se debe enviar un mail a la lista de docentes `algo3-doc` (y con copia a todo el resto del grupo) con el subject "Entrega TP2 2C2022". El cuerpo del mail debe contener el nombre y libreta de las **cuatro** personas que integren el grupo, más el link al perfil de usuario de UVa. El mail debe tener como adjuntos el pdf del informe y un archivo zip con el código de la solución de los tres problemas. El nombre de estos archivos debe ser el de los apellidos de las personas integrantes del grupo separados por guiones.

Ejercicio 1: *DFS*

Se pide resolver el problema 12363 de UVa llamado *Hedge Mazes*.[3] Se provee una traducción de su enunciado:

> 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

[3] https://onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=278&page=show_problem&problem=3785

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.

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 -

Se pide:

1. Diseñar un algoritmo que realice un preprocesamiento de complejidad $O(R+C)$ y luego responda cada consulta en $O(1)$. Este algoritmo debe pasar la prueba del juez de UVa.
2. Demostrar la correcitud del algoritmo.

Ejercicio 2: Árbol generador mínimo

Se pide resolver el problema 1265 de UVa llamado *Tour Belt*[4]. Se provee una traducción (resumida) de su enunciado:

La organizacion KTO (*Korea Tourism Organization*) quiere seleccionar un subconjunto de islas de un archipiélago para conformar un *Tour Belt*[5]. Para eso, calculó, para algunos pares

[4] https://onlinejudge.org/index.php?option=onlinejudge&Itemid=8&page=show_problem&problem=3706
[5] Un *Tour Belt* es simplemente un subconjunto de islas que se seleccionan para diseñar un programa turístico

de islas v, w, el *efecto de sinergia* $SE(u, v) = SE(v, u)$ que denota el valor turístico que tiene la presencia de ambas islas en simultáneo dentro del tour.

Sea $G = (V, E)$ el grafo pesado que tiene como nodos a las islas que considera la KTO, y donde hay un eje (v, w) con peso $SE(v, w)$ si el *efecto de sinergia* fue calculado para el par de islas v, w. Dado un subconjunto $B \subseteq V$ de islas habrá ejes que queden "dentro" del subconjunto y otros que queden "en el borde". Formalmente, los ejes internos a la componente B son aquellos vw tales que $v, w \in B$, mientras que los ejes del borde son aquellos vw tales que $v \in B$ y $w \notin B$ (o viceversa).

La KTO considera que un subconjunto B de islas es *candidato* a ser *Tour Belt* si $|B| \geq 2$, la componente es conexa y todos los ejes internos de la componente tienen peso mayor a los que están en el borde.

La KTO necesita de un programa que encuentre, dado un grafo pesado, la sumatoria de los tamaños de los subconjuntos candidatos.

Descripción de una instancia

La entrada posee múltiples casos de tests. La prímera lína contiene un entero T, que denota la cantidad de casos de tests. Luego, en las siguientes líneas, se encuentran cada uno de los T casos de test.

Cada uno comienza con una línea con dos enteros n y m, indicando respectivamente la cantidad de nodos y ejes del grafo, con $1 \leq n \leq 5000$ y $1 \leq m \leq n^2$. Los nodos están numeradas de 1 a n. Las siguientes m líneas contienen cada una 3 enteros v, w y k, donde k denota el peso de la arista vw.

Descripción de una salida

Para cada caso de test, se debe imprimir la sumatoria de los tamaños de los subconjuntos candidatos del grafo.

Entrada de ejemplo	Salida esperada de ejemplo
2 4 6 1 2 3 2 3 2 4 3 4 1 4 1 2 4 2 1 3 2 8 7 1 2 2 2 3 1 3 4 4 4 5 3 5 6 4 6 7 1 7 8 2	8 20

Se pide:

1. Diseñar un algoritmo de complejidad $O(m \log(n) + n^2 \alpha^{-1}(n))$[6] que resuelva el problema, donde α^{-1} denota la inversa de la función de Ackerman. Para lograrlo, considerar las siguientes ayudas:
   - Notar que si se ejecuta Kruskal para buscar un árbol generador máximo, toda componente *candidata* va a ser, en alguna iteración, una de las componentes conexas del bosque que mantiene el algoritmo.
2. Demostrar la correctitud del algoritmo.

Ejercicio 3: Camino mínimo

Se pide resolver el problema 1233 de UVa llamado *Usher*[7]. Se provee una traducción de su enunciado:

> En una iglesia de Nueva Zelanda, después de misa, el sacerdote pasa al portero una caja vacía que puede contener hasta c monedas. El portero luego le pasa la caja a cada uno de los feligreses, los cuales agregan algunas monedas, y la vuelven a pasar. Cada vez que la caja pasa por las manos del portero, este roba silenciosamente una de las monedas, la guarda en su bolsillo, y sigue pasando la caja.
>
> El comportamiento del portero está definido por un conjunto de feligreses a los que les puede pasar la caja. Por otro lado, el comportamiento de cada feligrés está definido por un conjunto de reglas, cada una consistiendo de una cantidad de monedas que coloca en la caja y la siguiente persona (feligrés o portero) al que ha de pasarle la caja. Cuando la caja se llena con c monedas, inmediatamente es devuelva al sacerdote, incluso en el caso en que un feligrés no haya terminado de colocar todas las monedas de la regla que estaba aplicando.
>
> El problema consiste entonces en determinar la ganancia máxima que puede obtener el portero, asumiendo que se puede controlar qué regla elige ejecutar cada feligrés (de las que tiene disponible) cuando le llega la caja.

Descripción de una instancia

> El input comienza con una línea que contiene la cantidad T de casos de test. Luego siguen los T casos de tests, y una línea final que contiene el número 0.
>
> La primera línea de cada caso de test consiste en dos números c y p, donde c indica la capacidad de la caja y p el número de feligreses. Se puede asumir que $1 \leq c \leq 10^7$ y $1 \leq p \leq 500$. La siguiente línea especifica el número q de feligreses a los cuales el portero puede pasar la caja. Los siguientes q enteros representan a cada uno de estos feligreses, los cuales están numerados de 1 a p. Cada línea i con $1 \leq i \leq p$ de las siguientes p líneas describe el comportamiento del feligrés i, y consiste de un conjunto de enteros separados por espacios. El primer entero especifica el número de reglas del feligrés. Cada regla está representada por un par de enteros, donde el primero denota el número de monedas a

[6] Conocemos mejores algoritmos para este problema. Por un lado, no es difícil remover el factor $\alpha^{-1}(n)$ que multiplica a n^2. Por otro, es incluso posible reducir ese término hasta hacerlo lineal.

[7] https://onlinejudge.org/index.php?option=onlinejudge&Itemid=8&page=show_problem&problem=3674

colocar en la caja (el cual siempre es mayor o igual a 2), y el segundo el feligrés al cual pasar la caja. En particular, si el segundo número es 0, entonces la caja se devuelve al portero. Se puede asumir que la cantidad de reglas para cada feligrés está acotada entre 1 y 1000, y puede ocurrir que dos reglas distintas definan el pasaje de la caja hacia un mismo feligrés.

Descripción de una salida

La salida consiste en una línea por cada caso de test. La i-ésima línea contiene la maxima cantidad de monedas que el portero puede obtener para el caso de test i.

Entrada de ejemplo	Salida esperada de ejemplo
1 10 2 2 1 2 2 6 0 4 2 1 5 0 0	2

Se pide:

1. Diseñar un algoritmo de complejidad $O(r \log(p))$ que resuelva el problema, donde p denota la cantidad de feligreses y r la cantidad total de reglas. Este algoritmo debe pasar la prueba del juez de UVa.
2. Demostrar la correctitud del algoritmo.

Ejercicio 4: Sistemas de restricciones de diferencias

Dado un sistema S de k ecuaciones sobre n variables $x_1 \dots x_n$, donde cada ecuación es de la forma $x_i - x_j \leq c$, y una lista D de m números enteros ordenados de menor a mayor, se quiere decidir si existe una asignación de las variables x_i a elementos en D que satisfaga las ecuaciones del sistema.

Descripción de una instancia

El input comienza con una línea que contiene la cantidad T de casos de tests. Luego siguen los T casos.

La primera línea de cada caso contiene 3 enteros k, n y m, separados por espacios, indicando respectivamente la cantidad de ecuaciones, la cantidad de variables, y el tamaño del conjunto D.

Las siguientes k líneas describen, cada una, una de las ecuaciones del sistema. Estas son descriptas por 3 números enteros a, b y c, donde a y b denotan índices de variables. Las variables están numeradas de 1 a n, por lo que $1 \leq a, b \leq n$. Los 3 números a, b y c en ese orden representan la ecuación $x_a - x_b \leq c$.

Después se sigue una línea con m números enteros $d_1 \dots d_m$ separados por espacios y ordenados de menor a mayor. Estos representan al conjunto D.

Descripción de una salida

La salida consiste en una línea por cada caso de test.

La i-ésima línea contiene el mensaje `insatisfacible` en caso de que el sistema de ecuaciones del i-ésimo caso de test no sea satisfacible asignando a cada variable x_i un valor dentro del conjunto D. Caso contrario, la i-ésima línea contiene n enteros representando una asignación que satisface el sistema. Más precisamente, si los enteros de la salida son $y_1 \dots y_n$ entonces debe pasar que asignando a cada variable x_i el valor d_{y_i} todas las ecuaciones del sistema se satisfagan.

Entrada de ejemplo	Salida esperada de ejemplo
3 2 3 3 2 1 -8 3 1 -5 2 5 10 3 4 3 1 2 0 3 4 0 1 3 0 3 6 9 1 2 3 2 1 -3 1 2 3	3 1 2 3 3 3 3 insatisfactible

Se pide:

1. Implementar el algoritmo de Fishburn [1] que resuelve el problema. La implementación debe tener una complejidad $O(kmn)$.
2. Contar las ideas fundamentales del algoritmo.

Referencias

[1] John P Fishburn. Solving a system of difference constraints with variables restricted to a finite set. *Information processing letters*, 82(3):143–144, 2002.