Trabajo Práctico 3

Fecha de entrega: viernes 24 de junio, hasta las 18:00 horas.

El doctor Emmet Brown está trabajando junto con Marty McFly en una nueva versión del condensador de flujo, que incorpora una nueva componente con química de avanzada. Sin embargo, no logran ponerse de acuerdo en cómo construirla. Por suerte, ambos coinciden en que el problema se resolvería si, dados dos grafos cualesquiera, pudieran encontrar un grafo que sea subgrafo de ambos y tenga la mayor cantidad de aristas posibles. Luego de investigar en la literatura, encontraron que el problema está estudiado como el *problema de máximo subgrafo común* (también conocido como *maximum common subgraph*, o *MCS*) y rescataron la siguiente formulación formal:

> Sean $G_1 = (V_1, E_1)$ y $G_2 = (V_2, E_2)$ dos grafos simples. El *problema de máximo subgrafo común* consiste en encontrar un grafo $H = (V_H, E_H)$ isomorfo tanto a un subgrafo[1] de G_1 como a un subgrafo de G_2 que maximice $\#E_H$.

Para ayudar a Marty y al Doc a viajar en el tiempo y resolver sus diferencias personales, en el presente trabajo práctico se pide lo siguiente.

1. El Doc ha estudiado este problema durante mucho tiempo, y cree entender que se puede resolver gracias a las aplicaciones prácticas de la teoría de grafos que describen en su paper Ehrlich y Rarey [1]. Está convencido de que con resolver MCS le alcanza para rediseñar correctamente la nueva componente en cuestión, pero Marty desconfía.

   a) Su tarea consiste en leer este paper y explicarle a Marty qué problema de grafos se buscar resolver en las aplicaciones del paper, encontrar las diferencias entre dicho problema y MCS, y explicar qué aplicaciones tiene este problema en la química. No es parte de este TP resolver el problema del paper, en todo caso es problema del Doc.

2. Para convencer a Marty, el Doc necesita mostrarle un algoritmo que efectivamente resuelva el problema. Dado que sus habilidades de programador están un poco oxidadas, se les pide diseñar e implementar un **algoritmo exacto** para MCS y desarrollar los siguientes puntos que justifiquen la solución encontrada:

   a) Explicar detalladamente el algoritmo implementado. Elaborar podas y estrategias que permitan mejorar los tiempos de resolución.

   b) Calcular el orden de complejidad temporal de peor caso del algoritmo.

   c) Realizar una experimentación que permita observar los tiempos de ejecución del algoritmo en función del tamaño de entrada y de las podas y/o estrategias implementadas.

3. El nuevo problema que tienen Marty y el Doc es que ahora no se ponen de acuerdo en qué estructura química deben utilizar para armar el condensador de flujos. Marty, a partir de las propiedades atómicas, ha obtenido una constante n y está seguro de que la solución que están buscando debe ser un grafo de a lo sumo n vértices (o como él lo dice, un subgrafo de algún K_n), mientras que el Doc ha estudiado las características moleculares e insiste con que debe ser un subgrafo de un cografo dado. Por las dudas, y para evitar más conflictos entre ellos, quieren cumplir con ambas especificaciones, siempre maximizando la cantidad de aristas, dado que éste es el punto clave para su funcionamiento dentro del condensador. Por eso les pedimos que los ayuden diseñando e implementando un **algoritmo exacto** para MCS que tenga **complejidad temporal polinomial** para el caso en el que G_1 **es un cografo y** G_2 **es un grafo completo** y desarrollen los siguientes puntos que avalen la solución encontrada (si estamos hablando de viajar en el tiempo, no hay margen de error posible):

   a) Explicar detalladamente el algoritmo implementado.

   b) Calcular el orden de complejidad temporal de peor caso del algoritmo.

[1] Un **grafo** $H = (V_H, E_H)$ es un *subgrafo* de $G = (V, E)$ si $V_H \subseteq V$ y $E_H \subseteq E$.

c) Realizar una experimentación que permita observar los tiempos de ejecución del algoritmo en función del tamaño de entrada.

4. Desgraciadamente, el problema es tan difícil que ni Marty ni el Doc conocen una manera de resolverlo en tiempo polinomial para el caso general. Para no quedarse con las manos vacías, Marty les pide diseñar e implementar una **heurística constructiva golosa** para MCS y desarrollar los siguientes puntos para conocer la calidad de las soluciones que le proveerán:

   a) Explicar detalladamente el algoritmo implementado.

   b) Calcular el orden de complejidad temporal de peor caso del algoritmo.

   c) Describir instancias de MCS para las cuales la heurística no proporciona una solución óptima. Indicar qué tan mala puede ser la solución obtenida respecto de la solución óptima.

   d) Realizar una experimentación que permita observar la performance del algoritmo en términos de tiempo de ejecución en función del tamaño de entrada.

5. Obviamente, el Doc está en desacuerdo y dice que la heurística golosa no es la mejor idea. Para justificar su postura les pide diseñar e implementar una **heurística de búsqueda local** para MCS , y desarrollar los siguientes puntos para, de una buena vez, convencer a Marty:

   a) Explicar detalladamente el algoritmo implementado. Plantear al menos dos vecindades distintas para la búsqueda.

   b) Calcular el orden de complejidad temporal de peor caso de una iteración del algoritmo de búsqueda local (para las vecindades planteadas). Si es posible, dar una cota superior para la cantidad de iteraciones de la heurística.

   c) Realizar una experimentación que permita observar la performance del algoritmo comparando los tiempos de ejecución y la calidad de las soluciones obtenidas, en función de las vecindades utilizadas y elegir, si es posible, la configuración que mejores resultados provea para el grupo de instancias utilizado.

6. A esta altura ya se cansaron de escuchar las peleas entre Marty y el Doc, con lo cual les dejaron muy en claro que lo último que les ofrecen es implementar una metaheurística para resolver el problema. Por suerte, fueron tomados en serio, así que evaluaron todos juntos la situación y decidieron que lo mejor es utilizar la **metaheurística Taboo Search**. Para ponerle punto final a esta historia, deberán diseñar e implementar un algoritmo para MCS que incluya la susodicha metaheurística, y desarrollar los siguientes puntos que validen la solución encontrada, para que finalmente reine el amor y la igualdad entre Marty y el Doc:

   a) Explicar detalladamente el algoritmo implementado. Plantear distintos criterios de parada y de tamaño de la lista taboo de la heurística.

   b) Realizar una experimentación que permita observar los tiempos de ejecución y la calidad de las soluciones obtenidas. Se debe experimentar variando los valores de los parámetros de la metaheurística (tamaño de la lista taboo, criterios de parada, etc.) y las vecindades utilizadas en la búsqueda local. Elegir, si es posible, la configuración que mejores resultados provea para el grupo de instancias utilizado.

7. Marty y el Doc son amigos de nuevo, pero les genera muchas dudas el no tener una garantía teórica de optimalidad. Para lo cual les piden, si fueran tan amables, que una vez elegidos los mejores valores de configuración para cada heurística implementada (si fue posible), realicen una **experimentación sobre un conjunto *nuevo* de instancias** para observar la performance de los métodos, comparando nuevamente la calidad de las soluciones obtenidas y los tiempos de ejecución en función del tamaño de entrada. Para los casos en los que sea posible, les encantaría comparar también los resultados contra los del algoritmo exacto.
   Ambos personajes son muy exigentes en cuanto a la representación de la información y la evidencia científica, con lo cual deben presentarles todos los resultados obtenidos mediante gráficos adecuados (u otras opciones que consideren provechosas) y discutir al respecto de los mismos.

8. Como ejercicio EXTRA y NO OBLIGATORIO pueden desafiar a Marty y el Doc, resolviendo el problema mediante un código en C++ que use cualquier técnica algorítmica o heurística que deseen pero que demore no más de 15 segundos en terminar. Este ejercicio debe ser resuelto en grupos de

a lo sumo dos personas (para no poner en desventaja numérica a nuestros personajes), que deben, a su vez, pertenecer al mismo grupo de TP. Todos los códigos que sean entregados como parte de este ejercicio competirán por un premio. Para la competencia se armará un fixture con los grupos participantes, donde cada etapa se resolverá corriendo el algoritmo de cada uno para algún(os) caso(s) de prueba y los grupos se irán eliminando por eliminación directa, hasta quedar un único grupo ganador. En cada instancia, cada grupo obtendrá un puntaje proporcional a la cantidad de ejes del resultado que se obtenga, siempre y cuando sea un subgrafo de ambos grafos, o 0 (cero) en caso contrario o si tarda más de 15 segundos en terminar. Los inscripción a esta competencia cierra el 10 de junio y se realizará enviando un correo electrónico a algo3.dc@gmail.com con subject COMPETENCIA-TP3, informando en el contenido nombre y apellido de cada uno de los integrantes del grupo. Hay interesantísimos premios y menciones para el(los) grupo(s) ganadores.

Condiciones de entrega y términos de aprobación

Este trabajo práctico consta de varias partes y para su aprobación se requiere aprobarlas todas. La nota final será un promedio ponderado de las notas finales de cada una, considerándose aprobados los trabajos con una nota de 5 (*cinco*) puntos o superior. De ser necesario (o si el grupo lo desea) el trabajo podrá reentregarse una vez corregido por los docentes y en ese caso la reentrega deberá estar acompañada por un *informe de modificaciones*. Este informe deberá detallar brevemente las diferencias entre las dos entregas, especificando los cambios, agregados y/o partes eliminadas del trabajo. Cualquier cambio que no se encuentre en dicho informe podrá no ser tenido en cuenta en la corrección de la reentrega.

Respecto de las implementaciones, se acepta cualquier lenguaje que permita el cálculo de complejidades según la forma vista en la materia. Además, debe poder compilarse y ejecutarse correctamente en las máquinas de los laboratorios del Departamento de Computación. La cátedra recomienda fuertemente el uso de C++, permitiendo también el uso de Java o Python, mientras que se debe consultar con los docentes la elección de otros lenguajes para la implementación.

Deberá entregarse un informe impreso que desarrolle los puntos mencionados en el enunciado. Por otro lado, deberá entregarse el mismo informe en formato digital acompañado de los códigos fuentes desarrollados e instrucciones de compilación, de ser necesarias. Estos archivos deberán enviarse a la dirección `algo3.dc@gmail.com` con el asunto *"TP 3: Apellido_1, ..., Apellido_n"*, donde n es la cantidad de integrantes del grupo y *Apellido_i* es el apellido del i-ésimo integrante.

La entrada y salida de los programas **deberá hacerse por medio de la entrada y salida estándar del sistema**. No se deben considerar los tiempos de lectura/escritura al medir los tiempos de ejecución de los programas implementados.

Formato de entrada: La entrada comienza con una línea con cuatro valores enteros, n_1 y m_1, n_2 y m_2, separados por espacios (con $n_i > 0$ y $m_i \geq 0$ para $1 \leq i \leq 2$); los valores n_i y m_i indican la cantidad de vértices y aristas de G_i, respectivamente. A continuación, siguen m_1 líneas, representando las aristas del grafo G_1 y luego m_2 líneas representando las aristas del grafo G_2. Cada una de estas líneas tiene el formato:

```
u v
```

donde `u` y `v` son los extremos de la arista representada (numerados de 0 a $n_i - 1$). Se puede suponer que los grafos son simples (i.e., sin bucles ni aristas repetidas).

Formato de salida: La salida debe contener varias líneas con el siguiente formato:

```
T E
v(1,0) v(1,1) ... v(1,T-1)
v(2,0) v(2,1) ... v(2,T-1)
a1 b1
a2 b2
...
aE bE
```

donde `T` es la cantidad de vértices del máximo subgrafo común H entre los dos grafos G_1 y G_2 y `E` la cantidad de aristas de dicho grafo H. `v(1,i)` es el vértice de H que se corresponde en el isomorfismo con el i-ésimo vértice de G_1, mientras que `v(2,i)` corresponde al i-ésimo vértice de G_2. Las `E` aristas del grafo H serán las aristas que conecten los vértices `ai` con `bi`

En el caso de los algoritmos exactos, de haber más de una solución óptima, el algoritmo puede devolver cualquiera de ellas.

Referencias

[1] HC. Ehrlich and M. Rarey Maximum common subgraph isomorphism algorithms and their applications in molecular science: a review. *Wiley Interdisciplinary Reviews:* Computational Molecular Science 1 (1), 68-79.