Problemas y algoritmos

Gráficas alcanzables

Una gráfica es una estructura de datos que nos permite representar un problema a través de un conjunto V, correspondiente a un conjunto de vértices y un conjunto E de aristas.
Como hemos visto en el curso, varios problemas pueden ser aplicados a esta estructura, en particular, uno de esos problemas es saber si dados dos vértices v1 y v2, es posible encontrar un camino desde el vértice v1 al vértice v2.
Este es un problema considerado de alcanzabilidad en el sentido de saber si un vértice puede ser alcanzado por otro.
Como hemos visto en el curso, este problema se puede reducir a un problema de decisión, en donde decimos *sí* en caso de que un vértice sea alcanzable por otro, de lo contrario decimos *no*.

Una forma de saber si un vértice es alcanzable por otro, es usando un algoritmo de búsqueda sobre gráficas. Estos algoritmos se basan en un sistema de marcado de nodos y un conjunto S. La idea es tomar el vértice de inicio y agregarlo a S. Por cada iteración se va a obtener un elemento del conjunto S y se van a obtener todos los vecinos del vértice.

Sin pérdida de generalidad supongamos que un vecino del nodo es i. Si S no está marcado entonces se marca y se agrega a S, en caso de que S ya sea un vértice marcado, no se realiza alguna operación y se sigue verificando con los vecinos del nodo tomado de S.
Esta iteración se realiza hasta que S sea vacío. En general, todos los nodos marcados, corresponden a los vértices alcanzables desde el vértice inicial. Si el nodo final, que en particular se quiere alcanzar, está marcado, decimos que sí existe una forma de alcanzar un vértice final desde el vértice inicial, de lo contrario, podemos responder que no son vértices alcanzables.
Es importante mencionar que la forma en que se van a marcar los vértices depende de la representación de una gráfica. Ya que una gráfica se puede representar con una matriz de adyacencias, la complejidad de acceso a los elementos de la estructura es constante.
Otro factor importante sobre el estilo de búsqueda es el criterio que tomemos para elegir un elemento del conjunto S para realizar una iteración. Si tomamos como criterio el tomar el vértice que ha estado más tiempo en S, el recorrido tendrá la forma de una búsqueda en anchura. Por el contrario si obtenemos los elementos de

S por el comportamiento de una pila, entonces el recorrido se vería como una búsqueda por profundidad. En el peor caso, tenemos n vértices que son adyacentes al resto de los vértices. En ese caso se tendrían que realizar n-1 verificaciones por cada vértice. Esto nos daría $O(n^2)$

Algoritmos polinomiales

Los algoritmos polinomiales son aquellos en los que se hace la intuición de que son factibles en poner en práctica respecto al tiempo en que un algoritmo puede regresar un resultado. Es importante decir que al referirnos a algoritmos polinomiales no estamos hablando exactamente de eficiencia, ya que encontrar algoritmos que son eficientes y no son polinomiales, y algoritmos polinomiales que son eficientes en la práctica.
Un hecho importante a destacar es que el tiempo que un algoritmo exponencial se tarda es tanto que podemos decir que su uso es casi insignificante en la práctica y poco útil, ya que sólo funciona para entradas muy limitadas, pero no para el promedio de las entradas con el que nos gustaría obtener un resultado en un tiempo considerablemente bueno. Sin embargo, en la práctica no podemos saber con certeza qué entrada le van

a pasar a un algoritmo.
Para estar seguros, de alguna forma sobre el tiempo que un algoritmo se puede tardar con la mayor cantidad de entradas, es considerar el peor caso posible con el que un algoritmo tendría que lidiar tanto en tiempo como en memoria y estos no necesariamente se cumplen con el mismo ejemplar del problema.

Flujo máximo

Para este problema se usa una gráfica conocida como red, la cual tiene los siguientes componentes: una gráfica $G = (V, E)$; S es un vértice que provee recursos; t es un vértice que representa un sumidero; y c es un conjunto $c(i, j)$ que representa la capacidad entre la arista i y la arista j. Un flujo, finalmente, se define como un valor de la cantidad de flujo que puede llegar desde un vértice inicial hasta un vértice final, usando la mayor capacidad de cada $c(i, j)$. El problema del máximo flujo consiste en encontrar el flujo con el valor más alto en recursos que se pueden transmitir. Como se puede notar, el problema del flujo máximo no es un problema de decisión, ya que un resultado del algoritmo no responde sí o no. Sin embargo, este problema se puede reducir a un problema de decisión

tomando una k y preguntando si existe un flujo, tal que sea menor a k. En este caso, el algoritmo ya nos respondería Si o no, lo cual lo convierte en un problema de decisión.
Este problema lo podemos ver como resolver una instancia del problema de alcanzabilidad. Ya que desde un vértice inicial (el vértice 1), podemos hacer un recorrido, pero llevando una variable que nos lleve la cuenta de cuál recorrido posible desde el recurso hasta el nodo más lejano (suponiendo que se encuentra al final del recorrido), para finalmente ver cuál fue la suma con mayor valor al recorrer todos los vértices.
El algoritmo de alcanzabilidad está dado $O(n^2)$. Pero en este caso extenderemos el problema a tener una variable C, que corresponde a la máxima capacidad de alguna arista en la red y de nC etapas. Este tiempo estará cambiando $O(n^2)$ por cada iteración. Así que por cada iteración se estará cambiando nCn^2 operaciones, lo que nos da como complejidad en tiempo $O(n^3C)$.
Sin embargo, el que nuestro algoritmo sea cúbico da cierto problema en cuestión de eficiencia. Ahora bien, el algoritmo no solo depende de n, sino también de C, que recordemos, es la máxima capacidad de cualquier arista en la red. Sin embargo, nos encontramos con otro problema respecto a n que afecta de forma terrible la

complejidad del algoritmo.
Suponiendo que siempre se elige la arista más corta para aumentar el flujo, las aristas de N y SN regresas se pueden convertir en un cuello de botella, el cual ya no dejará fluir los recursos de forma eficiente. A lo más podemos tener n^2 vértices y si por cada iteración del algoritmo encontramos un cuello de botella, el número de iteraciones estaría acotado por n^2, así que finalmente nuestro algoritmo se ve afectado directamente por n en tiempo $O(n^5)$.

Máquina biparbita

Una máquina biparbita trabaja, en particular, sobre una gráfica biparbita. Una gráfica biparbita G se compone de tres elementos; un conjunto U llamado chicos, un conjunto V llamado chicas; y un conjunto G de aristas de la forma $U \times V$.
Una máquina perfecta es un conjunto de aristas de una gráfica biparbita, tal que a cada elemento de V le corresponde uno y sólo un elemento de U. Esto es básicamente asignar a cada chico, una chica diferente.
En este momento se introduce el concepto de reducción, en el que una instancia de un problema A se puede resolver de forma equivalente resolviendo una instancia

de un problema B.
En este caso, podemos reducir el problema de una máquina perfecta en una gráfica bipartita con el problema del máximo flujo sobre gráficas. La forma en que se puede hacer es tomando una gráfica bipartita y aplicarle el algoritmo del máximo flujo. La gráfica es una máquina perfecta si tiene un flujo de valor n. El algoritmo del flujo máximo es polinomial, y una conversión de un problema a otro se hace de forma eficiente, así que el problema de encontrar una máquina perfecta se puede realizar en tiempo polinomial con la reducción de un problema a otro.
En general, el problema de reducción es muy importante porque si podemos y conocemos una forma de resolver un problema a partir de un algoritmo que garantiza resolver un problema equivalente, entonces podemos hacer una reducción para resolverlo, un problema con algoritmos ya hechos para problemas ya resueltos

Problema del Agente Viajero

Este problema consiste en encontrar el camino con menor costo entre n ciudades, sin repetirlas.
Una forma de poder resolver este problema es encontrar todas las combinaciones posibles y después verificar cuál de

todas esas posibles combinaciones es aquella que cuenta con el menor peso en el recorrido.
Si tenemos n ciudades como entrada al algoritmo, podríamos suponer que tenemos n posibles comienzos para el recorrido. Para elegir la segunda ciudad del recorrido podemos elegir entre n-1 posibilidades, para la tercera ciudad tendríamos n-2 posibilidades. De tal forma que obtendríamos $n(n-1)(n-2)\dots(1) = n!$ posibilidades.
El encontrar todas las combinaciones nos costaría $O(n!)$
Es una complejidad terrible que está fuera de los rangos de un problema polinomial. De hecho, este problema es un problema no determinista polinomial, ya que no existe una mejor forma de resolverlo.
Aunque existen heurísticas y métodos para obtener un resultado, es importante destacar que son aproximaciones al resultado correcto correspondiente a la solución más eficiente, pero un acercamiento a la mejor solución.