Proyecto #3

Diseño y análisis de algoritmos - Universidad de los Andes

Ronald Yesid Diaz Pardo - 202111309 - r.diazp, Nicolas Diaz Montaña - 202021006 - n.diaz9, Andres Felipe Guerrero - 202015143 - a.guerreros

Explicación de algoritmo para solucionar el problema

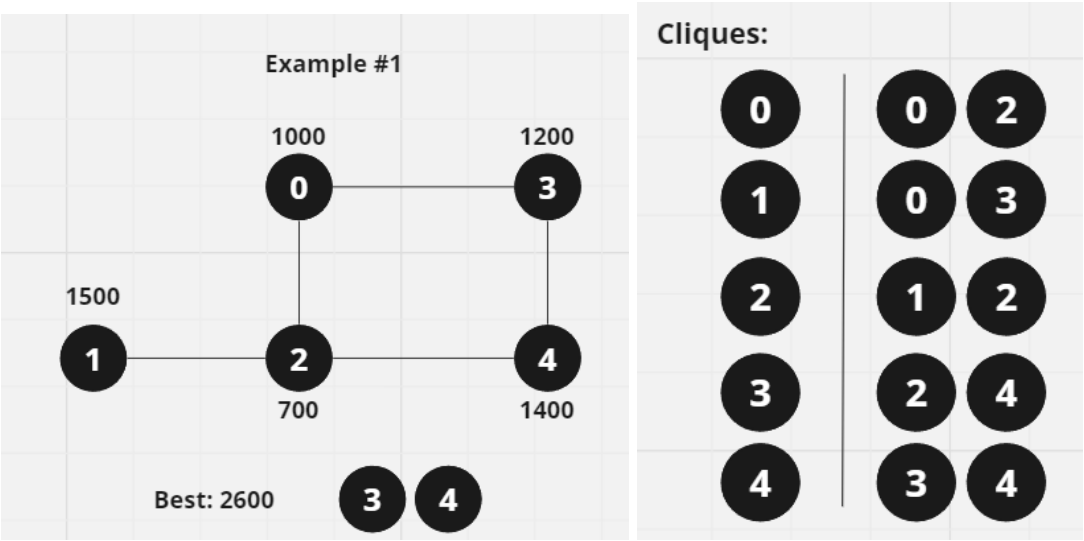
Para las soluciones propuestas se tomaron las entradas que se dan y se transformaron en variables que sirvan para solucionar el problema. La primera línea de la entrada, que representa la cantidad de dinero que aporta cada una de las personas interesadas, se convierte en una lista con la que se podrá encontrar el máximo de aportaciones. Ahora, las n líneas restantes, que indican cada persona los índices de las personas que conoce, se volverá una lista de adyacencia que será la representación del grafo. Siendo cada nodo una de las líneas, por ejemplo la primera línea el índice [0] de la lista de adyacencia y su conexiones serán el contenido de la línea .

a. Solucion greedy

Para la solución que denominamos “Greedy”, la cual es nuestras solución principal, lo que se hace en el código es elegir el vértice con el peso mayor y se recorre sus vecinos, de los cuales solo se agrega a un vecino si este conoce a todos los vértices de la solución actual (personas), es decir, si existe un edge. Al final, se retorna la estructura de personas o vértices elegidos, que forman un clique. Donde un clique es un grafo donde cualquier vértice u del clique conoce a cualquier otro vértice v del clique. Por lo tanto, se está seleccionando a personas que se conozcan entre ellos y los que pueden aportar más, aunque pueden haber casos donde la primera persona seleccionada no haga parte del clique máximo, lo cual resulta en que el algoritmo caiga en un bias que lo conlleve a escoger el clique equivocado, aun así el resultado depende en gran parte de la manera en la cantidad de dinero que aporta cada persona y la distribución de las conexiones de grafo.

b. Solucion cliques

Esta solución que denominamos “clique” es una solución directa al problema y que por lo tanto es 1-aproximado, lo cual implica tiempos de respuestas que suelen ser más altos que greedy.py. Esta solución busca hallar todos los cliques inicialmente de manera eficiente con memoria por medio del uso de generadores, tómese cliques como la definición dada en el inciso “a”, para luego recogerlos y escoger el que retorne una suma de mayor más alta que los demás. Esta búsqueda de cliques se hace por medio de Breadth First Search, para la cual se inicializan a todos los nodos como cliques individuales, luego para cada clique en cola se examinan los vecinos en común de los nodos en el clique; para cada vecino común, se crea un nuevo clique que incluye ese vecino y se agrega a la cola, se repite todo este proceso hasta que la cola está vacía.



Análisis de complejidad temporal y espacial

Greedy

La complejidad temporal del algoritmo es O(V^2) en el peor de los casos. Esto se debe a que para cada vértice, el algoritmo puede tener que iterar sobre todos los demás vértices para verificar si agregar un nuevo vértice al clique resultaría en un no-clique.

La complejidad espacial del algoritmo es O(V + E). El algoritmo mantiene una lista de vértices restantes (lo que requiere espacio O(V)) y una representación de gráfico que contiene todas las aristas (lo que requiere espacio O(E)).

Cliques

La complejidad temporal del algoritmo es O(V * d^3) = O(V*(V-1)^3) = O(V^4), donde V es el número de nodos y d es el grado máximo en el gráfico. Esto se debe a que para cada nodo, el algoritmo puede tener que iterar sobre todos los vecinos de ese nodo (lo que toma tiempo proporcional al grado del nodo) y para cada par de vecinos, puede tener que buscar en la lista de vecinos para encontrar vecinos comunes (lo que toma tiempo proporcional al grado del nodo al cuadrado). Luego, se sabe que el número máximo de edges que tendrá cada vértice es V-1, por lo que en el peor caso es V-1, lo cual puede ser simplificado a V^4 por la notación Big O.

La complejidad espacial del algoritmo es O(V + E), donde V es el número de nodos y E es el número de aristas. Esto se debe a que el algoritmo mantiene una cola de cliques para explorar, y cada clique puede contener hasta V nodos. Además, el algoritmo mantiene una lista de vecinos para cada nodo, que en total puede contener hasta 2E aristas (ya que cada arista se cuenta dos veces, una vez para cada nodo).

Respuestas a escenarios de comprensión de problemas algorítmicos

Escenario #1: Las personas podrían aceptar no conocer máximo una de las personas que van a aportar.

¿Qué cambios -si aplica- le tendría que realizar a su solución para que se adapte a este nuevo escenario?

En lugar de exigir que cada persona conozca a todas las demás, se podría relajar la restricción dentro del código para permitir que exista al menos una conexión directa entre la persona seleccionada y la que no conoce a los demás. Eso significa que ya no es necesario que se selecciones exclusivamente a los vecinos que conozcan a todos los demás vértices de la solución actual en el que se encuentre la iteración. Esto permitiría una mayor inclusividad y podría resultar en una formación de cliques práctica y un aporte de dinero mayor.

Aun así esto puede traer retos, porque ahora se tiene que tener en cuenta que solamente los donantes van a aceptar 1 y solo 1 persona que no conozcan. Esto implica que se debe llevar un registro adicional, donde se tiene que saber qué nodos ya han aceptado o no a 1 persona que no esté conectada a ellas. Y esto, dependiendo del caso, puede disminuir la eficiencia del algoritmo. Ya que se agrega una nueva condición de agregado.

¿Qué cambios -si aplica- le tendría que realizar a su solución para que se adapte a este nuevo escenario?

Para los cambios que pueden hacerse en nuestro código son dos: (1) Agregar una forma de llevar registro de si un nodo ya aceptó no conocer una por medio de una lista de n elementos donde se tenga booleanos y cada índice sea un nodo. (2) Agregar la condición de aceptar a un nodo “desconocido” si en la lista anterior es un False.

Escenario #2: Las personas solamente necesitan conocer al menos una de las personas que van a aportar.

¿Qué nuevos retos presupone este nuevo escenario?

el escenario 2 ya no es problema de clique, sino de componentes conectados. En lugar de exigir que cada persona conozca a todas las demás, podemos cambiar esa restricción y simplemente ahora las personas solamente necesitan conocer al menos una de las personas que van a aportar.por lo que se elegirá a la persona que más aporta y elegir el amigo que más aporta, verificar si conoce a alguien que esté en la lista de los que aportan, en caso que si, se agrega y se continúa con el siguiente amigo que más aporte y así sucesivamente hasta que lleguemos a una persona que, o no tenga amigos o todos sus amigos ya están en la lista, al hacer esta restricción más flexible que el escenario original, se puede crear un algoritmo óptimo y rápido por lo que no es necesario un algoritmo aproximado en este caso.

¿Qué cambios -si aplica- le tendría que realizar a su solución para que se adapte a este nuevo escenario?

Realmente se puede realizar una nueva implementación usando algún tipo de algoritmo que halle los componentes conectados.para estos problemas se conocen algoritmos que con capaces de obtenerlos en O(V) como es el caso de un BFS, por lo cual no son complejos computacionalmente y se obtendrá siempre la respuesta mas optima.