Trabajo Práctico

Sebastián Cherny y Martín Vacas Vignolo

3 de julio de 2019

1. Espíritu Navideño

1.1. Modelado

Vamos a modelar el shopping como una grafo, donde cada casilla es un nodo, y hay una aristas entre dos nodos si y sólo si se puede ir de uno hacia otro. Se ve que en esta analogía, un nodo puede estar bloqueado ('X', no puede ser atravesado), libre ('.'), puede contener a un local ('L') o a un comprador ('C'). En un mismo nodo puede haber más de un comprador a la vez, excepto por el momento inicial según detalla el problema.

Entonces, que un un comprador vaya a un local es lo único que hay que modelar, que va a ser simplemente un camino en el grafo que comienza en el nodo en el que está el comprador, no atraviesa ningún nodo bloqueado, y termina en el local deseado.

1.2. Poniendose de acuerdo

Para esta parte, queremos averiguar la menor cantidad de tiempo en la que el último comprador obtiene su regalo. Para esto, lo que vamos a emplear es una búsqueda binaria en el tiempo. Cómo sabemos si, para un tiempo T fijo, los compradores pueden todos obtener sus regalos? Lo que hacemos será pensar que el comprador C_i va a querer ir al local L_j si y sólo si están a menos de (o exactamente igual a) T segundos de tiempo de caminata entre ellos, es decir, como se mueven a razón de una casilla por segundo, que en el grafo se traduce a moverse una arista por segundo, si y sólo si están a menos de distancia T.

Una vez que sabemos a qué locales puede ir cada comprador, no nos importa a cuáles vayan, ya que queremos saber si se puede, y yendo a cualquiera de estos tardará un tiempo menor o igual a T por cómo definimos los locales a los que puede ir en este paso. Obviamente, como se van a poner de acuerdo al principio, no va a pasar que un comprador vaya a un local a comprar su regalo y éste no tenga producto porque llegó otro antes; si se ponen de acuerdo al principio adónde va cada comprador, van directamente lo más rapido posible hacia allí.

Entonces, lo que tenemos en el paso de la binaria podemos pensarlo como un grafo bipartito, donde de un lado están los compradores y del otro los locales, y hay una arista entre dos nodos si y sólo si el comprador tarda un tiempo menor o igual a T en llegar al local. Y la respuesta a "todos pueden comprar sus regalos"la dará un algoritmo que diga si hay un matching completo (para los compradores) o no. Si lo hay, es que podemos seleccionar aristas de manera tal que los compradores vayan a los locales asignados, y por cómo pusimos las aristas en el grafo, no tardarán mas de T segundos y sabremos que con T segundos, seguro pueden obtener todos sus regalos. Si no hay un matching completo, es que esto no es posible, y sabemos que con T segundos no puede obtener todos sus regalos.

El único caso que hay que mirar, es si para ningún tiempo todos puede obtener sus regalos. Por ejemplo, si hay un comprador con todas sus casillas vecinas bloqueadas, no habrá manera de que compre un regalo y la respuesta será -1. Esto lo sabremos al terminar la binaria, cuando hayamos visto que no pueden completar la tarea para ningún tiempo menor a $R * C$. Ese tiempo es lo máximo que le lleva a un comprador llegar a un local, pasando por todas las casillas (en realidad esa cota puede mejorarse ya que si no puede ir directo es porque hay casillas bloqueadas, y esas casillas no las recorrerá. Pero el orden de magnitud no cambiará).

1.2.1. Complejidad

Sea CC la cantidad de compradores, y CL la cantidad de locales.

Primero, haremos un precómputo de las distancias desde cada comprador a cada una de las otras casillas, guardando las distancias hasta las casillas con locales, que son las que nos importan. Esto lo hacemos simplemente con un bfs desde cada comprador. Este precómputo tiene complejidad $O(CC * R * C)$. La búsqueda binaria, como ya vimos, nos hará hacer $O(log_2(R * C))$ pasos. Y en cada binaria, lo que hacemos es recorrer los compradores, y para cada comprador recorremos las distancias guardadas hacia los compradores, y si es menor al tiempo de la binaria lo agregamos al grafo bipartito que usaremos. Armar el grafo entonces tiene complejidad $O(R * C)$, ya que agregar cosas a vector y comparar tiempos tienen complejidad constante.

Una vez obtenido el grafo, buscamos un matching. El algoritmo de esto es conocido y básicamente consiste en un DFS desde cada nodo de los compradores, por lo que su complejidad es $O(CC * cantidadDeAristas) = O(CC * (CC * CL))$.

Entonces, la complejidad para obtener el resultado de esta parte es $O(CC * R * C + log_2(R * C) * CC * CC * CL)$, que si las cantidades de locales están limitadas a a lo sumo 50, queda en $O(50 * 50 * 50 + log_2(50 * 50) * 50 * 50 * 50) = O(13 * 50 * 50 * 50) = O(1{,}6E6)$ que entra en tiempo.

1.3. Siendo egoístas

Para esta parte, no hay ningún mínimo ni máximo que buscar, es solamente simular los recorridos de los compradores y a partir de ahí, ver, para cada local, quién llega antes y quién debe ir a comprar a otro local.

Para esto entonces llamaremos **evento** a la llegada de un comprador a un local. Si tuviéramos la lista de eventos, con el momento en el que cada comprador llega a cada local (pasando lógicamente por locales previamente si no es el más cercano), entonces el problema se resuelve simplemente recorriendo los eventos, desde el primero que ocurre hasta el último, y si ese evento es la primera llegada a un local entre todos los compradores, como los eventos están ordenados por tiempo, no habrá otro comprador que llegue antes, por lo que el comprador en cuestión comprará el regalo ahí, y de ahí en más, este comprador y este local no nos interesan más, es decir no tendremos en cuenta eventos relacionados con este comprador y local; si bien es importante saber que el evento de llegada de otro comprador hasta acá sigue existiendo, por lo que este otro comprador deberá seguir recorriendo **desde este local**.

Para obtener los eventos de cada comprador, lo que hacemos primero un bfs hasta encontrar el primer local al que irá, y luego desde cada local al que va llegando, un bfs para encontrar el local más cercano a éste **que no visitamos**. Una primera idea podría ser "desde cada local averiguar el local más cercano a él"para que cuando un comprador llega, si no hay regalo, va al más cercano a éste. Pero en ese caso si en en el segundo local tampoco hay regalo, puede ocurrir que el más cercano a éste sea el primer local al que fue (aunque no necesariamente), y en ese caso nunca encontraríamos el primero local al que este comprador llega. Entonces, para cada comprador, tenemos que ver en qué momento llegará a cada local para tener así todos los eventos. Entonces para cada local, nos moveremos al local más cercano por el cual este comprador no pasó. Guardaremos la información de que el comprador ahora está en ese local, y buscaremos así el recorrido por todos los locales del comprador, realizando un bfs por cada local nuevo al que vaya el comprador, marcando los locales visitados, hasta que haya visitado todos.

Con el algoritmo descripto tendremos entonces todos los $O(CC * CL)$ eventos, que ordenamos por tiempo. Al tenerlos ordenados, como ya dijimos, los recorremos desde el primer evento que ocurre hasta el último, y si estamos en un evento en el cual el comprador todavía no compró y el local todavía no vendió, marcamos a la venta como ejecuta (el comprador compró y el local vendió), y continuamos recorriendo, guardando la información del tiempo, ya que lo que queremos es el momento en el que se ejecuta la última compra/venta.

1.3.1. Complejidad

Para cada comprador hacemos un bfs por cada vez que va a ir hacia un local, entonces la complejidad es $O(CC * CL * C * R)$. Luego tenemos los eventos, que serán $O(CC * CL)$, y los ordenamos, lo que tiene una complejidad de $O(CC * CL * log_2(CC * CL))$. Luego los recorremos y

vamos marcando qué compradores y locales participan de la compra/venta, que se hace en tiempo constante modificando variables de tipo bool, y guardamos el momento en el que se ejecuta la acción. La complejidad entonces queda de $O(CC * CL * C * R + CC * CL * log_2(CC * CL)) = O(50^4 + 50^2 * 13) = O(6E6)$ que entra en tiempo.

2. Juego de palabras

2.1. Juego encontrado

Libro: "Ozma of Oz" - http://www.gutenberg.org/cache/epub/486/pg486.txt
Palabras:

Oarless
Zabaione
Macduff
Adherent
Olenellus
Fierily
Outhire
Zoothomes

Frase del libro: "Come here, Billina, and I'll let you out; for Ozma of Oz is here, and has set us free."

2.2. Código

En vez de hacerlo con Simplex, realizamos un código en Python que hacía un backtracking. Lo que hace el código es primero obtener todas las frases del libro, y luego guardar lo que queremos que se forme con las primeras letras. En nuestro caso queríamos que se formara el nombre del libro.

Luego, con el método de Backtracking con Branch and Bound, el algoritmo intentaba formar una palabra que empiece con la letra que queremos (si estamos buscando la tercera palabra que empiece con la tercera letra del nombre del libro, por ejemplo), y que todas las letras de la palabra estén en la frase, borrando las letras que va usando en palabras. Lo que importa es que las letras estén en lo que nos quedó de la frase tantas veces como necesitemos en la palabra. Por ejemplo, si queremos la palabra "Oarless", necesitaremos que en la frase haya al menos 2 letras s. Luego, al buscar la palabra "Zabaione", necesitaremos que la a esté al menos 2 veces en la frase restante, es decir la frase que se obtiene de sacarle a la original, las letras de oarless tantas veces como aparezca cada letra.

El método de B&B se ve en que si al buscar una palabra, la i-ésima, tenemos pocas letras (queremos que todas las palabras tengan entre un mínimo y un máximo de cantidad de letras), o tenemos muchas letras, o no tenemos suficientes letras ni siquiera para cubrir las letras del nombre del libro que nos faltan poner como primeras letras de palabras, entonces instantáneamente dejamos de analizar la rama actual.

Para recorrer las palabras con una determinada primera letra, nos guardamos todas las palabras del diccionario (con longitud entre el mínimo y el máximo ya que si no cumplen eso sabemos que no las usaremos) en un mapa que tiene para cada letra, las palabras que empiezan con ella. Entonces es fácil acceder de una a todas las palabras que empiezan con una determinada letra.