Complejidad del Hungaro

Daniel Penazzi

27 de mayo de 2021

Tabla de Contenidos

Complejidad del algoritmo Húngaro

- Restar el minimo de cada fila y de cada columna es $O(n^2)$.
- Calcular el matching inicial de ceros es $O(n^2)$ (hay que revisar n filas y para cada una de ellas las n columnas)
- Para lo siguiente, hay una alta dependencia de la implementación particular que se haga.
- Recordemos que la siguiente parte consiste en encontrar un matching maximal de ceros, y si ese matching maximal no es perfecto, “cambiar la matriz”y volver a buscar matching maximal.
- Entonces, primero, dependeria de como implementemos el algoritmo de encontrar matching maximal.

Complejidad del algoritmo Húngaro versión Dinic

- La forma mas eficiente de hacer esta parte era usar Dinic en el network asociado, que por las carácterísticas especiales del network, tiene complejidad $O(n^{\frac{5}{2}})$.
- Pero esto no lleva a la mejor complejidad global del Húngaro.
- Pues si cuando encontramos el matching maximal, y no es perfecto, y tenemos que cambiar la matriz, y tenemos que recomenzar Dinic desde el principio, creando un network auxiliar, entonces tendriamos $O(n^{\frac{5}{2}})$ por cada vez que cambiemos la matriz.

Complejidad del algoritmo Húngaro versión Dinic

- Ademas en esta implementación no queda claro cuantas veces habria que hacerlo.
- Aún si valiera la cota $O(n^2)$ para el número total de cambios de matrices que se puede ver que vale para la implementación que vimos la última clase (lo cual no es claro para esta implementación con Dinic), entonces tendriamos complejidad total $O(n^{\frac{9}{2}})$ que no es la mejor posible.
- Quizas se pueda implementar con Dinic eficientemente haciendo algunos retoques, para lograr una complejidad comparable a la que daremos en esta primera sección, pero parece dudoso que se pueda modificar la implementación de Dinic para lograr la complejidad que veremos en la segunda sección. Al menos no conozco una.

Complejidad del algoritmo Húngaro como lo vimos la última clase

Teorema

La complejidad del algoritmo Húngaro como lo implementamos para correrlo en los ejercicios es $O(n^4)$.

- Prueba:
- De la forma en que lo implementamos la última vez, y con la cual haremos los ejercicios, corremos Edmonds-Karp sobre la matriz, y, crucialmente, no tenemos que cambiar ningún network auxiliar al cambiar la matriz, y vimos que al cambiar la matriz se pueden perder algunos ceros pero ninguno del matching parcial que tenemos hasta ese momento.

Complejidad del algoritmo Húngaro como lo vimos la última clase

- Como no se pierde ningún cero del matching parcial que tenemos, podemos continuar con la busqueda como veniamos, simplemente revisando las columnas donde hay nuevos ceros si lo hacemos completamente con el sistema de matriz o bien agregando esas columnas a la cola si lo hacemos con colas explícitas.
- De esta forma continuaremos con varios posibles cambios de matriz en el medio, pero terminaremos agregando UN lado extra al matching.
- Para completar un matching perfecto podemos tener que hacer esto un máximo de $O(n)$ veces (pues hay n lados en un matching perfecto).

Complejidad del algoritmo Húngaro como lo vimos la última clase

- Entonces como la cantidad de lados agregados al matching inicial es $O(n)$, la complejidad total del Húngaro es $O(n^2) + O(n)$.(complejidad de extender el matching en un lado).
- (El $O(n^2)$ es por por la resta de mínimos inicial mas matching inicial).
- Tenemos que ver entonces cual es la complejidad de extender el matching en UN lado.

Complejidad de extender el matching en un lado

- Si ignoramos por ahora la complejidad de cambiar la matriz, entonces el algoritmo para extender el matching en un lado es $O(m)$, pues es la busqueda de un camino aumentante con el BFS de Edmonds-Karp si lo hacemos con colas, lo cual seria $O(n^2)$ en estos casos pues estamos tomando matrices con $m = O(n^2)$ en general.
- o bien, si lo hacemos con matriz, tambien hay que ir revisando filas y columnas para etiquetearlas.
- Revisar las columnas, si uno construye bien la estructura de datos, será $O(1)$ pues al revisar la columna sólo queremos ver si esta “libre ”o no, y si no está libre, cuál es la fila matcheada con esa columna, pero obviamente esto lo podemos tener guardado de forma tal que revisarlo sea $O(1)$.

Complejidad de extender el matching en un lado

- Revisar las filas por otro lado es mas complicado pues tenemos que buscar a todos los vecinos, es decir, todos los ceros. Asi que revisar cada fila es $O(n)$.
- La complejidad total peor posible de tener que revisar todas las filas hasta poder extender el matching en un lado es entonces $O(n^2)$.
- La complejidad total del Húngaro con esta implementación, si ignoramos el costo de cambiar la matriz, seria entonces $O(n^2) + O(n).O(n^2) = O(n^2) + O(n^3) = O(n^3)$.
- Pero lamentablemente no podemos ignorar el costo de cambiar la matriz.

Agregando la complejidad de los cambios de matriz al cálculo.

- Si la complejidad de cambiar la matriz es CM y tenemos que hacer un máximo de T cambios de matriz para agregar un lado, entonces la complejidad no es $O(n^3)$ sino:

$$O(n^2) + O(n).\left(O(n^2) + CM.T\right) = O(n^3) + O(n).(CM.T)$$

- Asi que tenemos que calcular CM y T. Empecemos con CM.

Complejidad de un cambio de matriz.

- Cambiar la matriz requiere:
  1. Calcular m
  2. Restar m de S y sumarlo a $\Gamma(S)$
- Calcular m requiere calcular el mínimo de los elementos de $S \times \overline{\Gamma(S)}$ asi que es $O(n^2)$.
- Restar m de cada fila es $O(n)$ pues hay que restarlo a cada elemento.
- Por lo tanto restarlo de las filas de S es $O(n).|S| = O(n^2)$.

Complejidad de un cambio de matriz.

- Similarmente sumarle a las columnas de $\Gamma(S)$ es $O(n^2)$.
- Aún si usamos el truco de restar m solamente de $S \times \overline{\Gamma(S)}$ y sumar m solamente a $\overline{S} \times \Gamma(S)$, la complejidad queda $O(n^2)$.
- Por lo tanto la complejidad total CM de un cambio de matriz es $O(n^2) + O(n^2) = O(n^2)$
- Ahora acotemos T.
- Acotar T depende crucialmente del hecho que continuamos el algoritmo con el matching parcial que teniamos.

Propiedad Clave

Propiedad Clave

Luego de un cambio de matriz, o crece el matching o crece el S.

- Mas precisamente:

Propiedad Clave

Supongamos que el algoritmo de extensión de matching de ceros en una matriz C se detiene al encontrar un S con $|S| > |\Gamma(S)|$ y que cambiamos la matriz restando $m = min\{C_{x,v} : x \in S, v \in \overline{\Gamma(S)}\}$ de las filas de S y sumando m a $\Gamma(S)$, y luego continuamos la busqueda desde donde la habiamos dejado.
Entonces, con la nueva matriz obtenida, o bien al correr el algoritmo se agrega un nuevo lado al matching, o bien se detiene con un S_{nuevo} con $|S_{nuevo}| > |\Gamma(S_{nuevo})|$ tal que $|S_{nuevo}| > |S|$.

Prueba de la propiedad clave

- Prueba: Sea $x \in S$, $v \in \overline{\Gamma(S)}$ tal que $m = C_{x,v}$.
- Al restar m de S la nueva matriz tendrá un cero en la entrada x, v.
- Como $x \in S$, al continuar el algoritmo de busqueda, x agregará v a la cola.
- (sabemos que v no estaba antes en la cola pues $v \in \overline{\Gamma(S)}$).
- En terminos de la matriz, la fila x etiqueteará la columna v que antes no podia pues $C_{x,v} > 0$, y sabemos que v no estaba etiqueteada pues las únicas columnas etiqueteadas son las de $\Gamma(S)$ y tenemos que $v \in \overline{\Gamma(S)}$.

Prueba de la propiedad clave

- Ese v puede o no formar parte del matching parcial que estamos queriendo extender.
- Si no formase parte del matching, entonces $f(\overrightarrow{vt}) = 0$ y llegariamos a t, tendriamos un nuevo camino aumentante y podriamos extender el matching.
- Si lo estamos haciendo con la matriz, si v no formase parte del matching parcial, entonces cuando la etiqueteamos veria que esta "libre" y podriamos extender el matching.
- Entonces si v no forma parte del matching parcial, podemos extender el matching.
- ¿Qué pasa si v forma parte del matching parcial?

Prueba de la propiedad clave

- Entonces la columna v esta matcheada con alguna fila z.
- En particular v es vecino de z, y como $v \notin \Gamma(S)$, entonces z no puede estar en S.
- Pero al estar v etiqueteado y matcheado con z, entonces etiqueteamos z con la etiqueta "v".
- (si lo hacemos con colas, v agregaria a z a la cola).
- Es decir, agregamos z al "nuevo" S

Prueba de la propiedad clave y cota para T.

- Si luego de agregar a z y continuar la busqueda podemos extender el matching, todo bien, y si no, terminaremos con un nuevo S que tendrá a todos los elementos del S viejo, mas AL MENOS a z.
- asi que será estrictamente mas grande que el viejo S.
- Fin propiedad clave.

Corolario

$T \leq n$.

- Prueba: por la propiedad clave, como S no puede crecer mas que $O(n)$ veces pues hay sólo n filas, entonces luego de a lo sumo $O(n)$ "crecimientos de S" debemos si o si tener que poder extender el matching en al menos un lado.
- Asi que $T \leq n$.

Conclusión del teorema

- Habiamos visto que la complejidad del Húngaro era

$$O(n^3) + O(n).(CM.T)$$

- y calculamos que $CM = O(n^2)$ y $T \leq n$.
- Por lo tanto la complejidad queda:

$$\begin{aligned} O(n^3) + O(n).(CM.T) &= O(n^3) + O(n).\left(O(n^2).n\right) \\ &= O(n^3) + O(n).\left(O(n^3)\right) \\ &= O(n^3) + O(n^4) = O(n^4) \end{aligned}$$

Otra implementación

- La implementación que dimos, que es la original de Kuhn, la dimos asi porque es la mas fácil para implementar a mano.
- Pero esa complejidad puede mejorarse con algunos trucos en la implementación.
- Sólo que sólo es realmente mejor en una computadora, bajo una cierta suposición, y no a mano.
- Esta implementación usa directamente la implementación matricial completa, sin creación de colas.
- Y usa varios “trucos” que sólo tienen sentido hacer en una computadora, no a mano.

Teorema principal de la sección

Teorema

Bajo ciertas suposiciones razonables, el algoritmo Húngaro puede implementarse en $O(n^3)$.

- Prueba:
- La idea es mirar mas en detalle CM, la complejidad de cambiar la matriz.
- Vimos que la complejidad del Húngaro es

$$O(n^3) + O(n).(CM.T)$$

  y que $T \leq n$.
- Asi que la complejidad es

$$O(n^3) + O(n).(CM.n) = O(n^3) + O(n^2).(CM)$$

Prueba del teorema

- En la versión que hacemos a mano, y que tambien se puede implementar fácilmente en una computadora, $CM = O(n^2)$ y probamos que de ahi sale la complejidad total $O(n^4)$.
- Pero si pudieramos implementar un cambio de matriz en tiempo $O(n)$ entonces tendriamos

$$O(n^3) + O(n^2).O(n) = O(n^3) + O(n^3) = O(n^3)$$

- y probariamos el teorema.
- A primera vista esto parece imposible.

Cambiando la matriz en tiempo $O(n)$

- Recordemos que para cambiar la matriz debemos hacer dos cosas:
  1. Calcular el m.
     - que es un mínimo sobre una cantidad de entradas que son potencialmente $O(n^2)$.
  2. Restar m de algunas filas y sumarlo a algunas columnas.
     - todo lo cual es restar y sumar sobre una cantidad posible de $O(n^2)$ entradas.
- Asi que el sentido común diria que no es posible hacer [1] y [2] en tiempo $O(n)$.
- Pero si se puede, con algunos trucos interesantes.

Calculando m en tiempo $O(n)$

- Primero veamos como se puede hacer lo aparentemente imposible de calcular un mínimo de $O(n^2)$ elementos en $O(n)$.
- En realidad estrictamente hablando, no se puede...pero esencialmente si, con 2 trucos.
- El primer truco que se usa es que mínimos de mínimos es mínimo.

$$\begin{aligned} m &= \text{mínimo de los elementos que estan en las filas de } S \\ &\quad \text{y columnas de } \overline{\Gamma(S)} \\ &= \min\{C_{x,v} : x \in S, v \in \overline{\Gamma(S)}\} \\ &= \min\{\min(\{C_{x,v} : x \in S\}) : v \in \overline{\Gamma(S)}\} \end{aligned}$$

Calculando m en tiempo $O(n)$

- Sea $mS[v] = \min\{C_{x,v} : x \in S\}$.
- Entonces lo anterior dice que

$$\begin{aligned} m &= \min\{\min(\{C_{x,v} : x \in S\}) : v \in \overline{\Gamma(S)}\} \\ &= \min\{mS[v] : v \in \overline{\Gamma(S)}\} \end{aligned}$$

- Asi que si definimos un array mS tal que para cada columna $mS[v]$ indica el mínimo valor de los elementos de la columna v que esten en filas de S: $mS[v] = \min\{C_{x,v} : x \in S\}$
- podemos calcular $m = \min\{mS[v] : v \in \overline{\Gamma(S)}\}$ en tiempo $O(n)$.

Calculando mS eficientemente

- Pero aca parece que se esta haciendo trampa:
- si TENGO los $mS[v]$ puedo calcular m en tiempo $O(n)$, pero calcular cada $mS[v]$ lleva tiempo $O(n)$ y hacerlo para todos sería $O(n^2)$.
- ¿O no?
- Bueno, si, calcularlos lleva tiempo total $O(n^2)$ y por eso dije antes que en realidad no podemos calcular un mínimo de $O(n^2)$ elementos en $O(n)$....pero en realidad si pues podemos “ocultar” este costo.
- El truco esta en “repartir” ese costo $O(n^2)$ en otras partes del algoritmo que de todos modos van a demorar $O(n^2)$!!

Calculando eficientemente los mS

- Recordemos que cada vez que una fila se agrega a S luego en algún momento hay que escanear toda la fila para encontrar los vecinos, buscando los ceros de esa fila, para poder etiquetear las columnas correspondientes.
- Este escaneo de cada fila es $O(n)$ para cada fila y $O(n^2)$ total.
- Ahi estamos perdiendo tiempo con cada uno de los elementos de la fila que no son ceros, pues los tenemos que mirar igual pero no contribuyen en nada al etiqueteado de nuevas columnas.
- ¿Que tal si aprovechamos que los tenemos que mirar igual para usarlos en otra cosa?

Calculando eficientemente los mS

- Si inicializamos los $mS[v]$ a un número muy grande y luego, cada vez que revisamos una fila x de S para buscar vecinos:
  - Aprovechamos para cada elemento que este en la fila x, columna v le miramos el valor $C_{x,v}$, que tenemos que mirar de todos modos para saber si es cero.
  - Lo comparamos con lo que tiene $mS[v]$
  - y si $C_{x,v} < mS[v]$ actualizamos $mS[v]$ y lo volvemos igual a $C_{x,v}$.
- entonces en todo momento $mS[v]$ tendrá efectivamente el mínimo de los elementos de la columna v en las filas de S.
- Esta actualización es $O(1)$ y se hace mientras escaneamos la fila x de S buscando vecinos, asi que no agrega nada esencial al costo global.
- Gracias a este “truco sucio” podemos lograr lo aparentemente imposible.

Otra ventaja de mS

- $mS[v]$ tiene otra ventaja para calcular m.
- Vamos a calcular $m = \min\{mS[v] : v \in \overline{\Gamma(S)}\}$ pero
- ¿cómo sabemos eficientemente cuando una columna v está o no en $\overline{\Gamma(S)}$?
- Hay varias formas de implementar esto, pero teniendo los $mS[v]$, hay una muy fácil.
- $\Gamma(S)$ son los vecinos de S, es decir, aquellas columnas tal que en alguna fila de S tienen un cero.
- Por lo tanto, $mS[v] = 0$ si $v \in \Gamma(S)$.
- Asi que para calcular $m = \min\{mS[v] : v \in \overline{\Gamma(S)}\}$ en realidad simplemente hacemos

$$m = \min\{mS[v] : mS[v] \neq 0\}$$

Restando y sumando m en tiempo $O(n)$

- Nos queda ahora como restar m a S y sumarlo a $\Gamma(S)$ en tiempo $O(n)$.
- Esto es mucho mas fácil, pero requiere una suposición que de todos modos es razonable.
- En vez de restar/sumar a cada elemento de la matriz, lo restamos/sumamos a la fila/columna en abstracto
- Usamos un array, digamos RF tal que $RF[x]$ indica cuanto hay que restarle a los elementos de la fila x.
- Y otro array, digamos SC tal que $SC[v]$ indica cuanto hay que sumarle a los elementos de la columna v
- En vez de cambiar todos los elementos, simplemente actualizamos los arrays RF y SC, y esto se puede hacer en tiempo $O(n)$.

Restando m en tiempo $O(n)$

- Pero, otra vez, esto parece magia negra que no va a funcionar:
- Esta bien, dejamos indicado cuanto deberiamos restar/sumar, pero si nunca restamos/sumamos la matriz no se actualiza y entonces ¿cómo buscamos ceros?
- Bueno, ahi hay un truco que depende de la capacidad de hacer operaciones aritméticas.
- Lo que hacemos es que, cuando estamos buscando ceros, en vez de chequear:
  - if(0== C[x][v])....
- debemos hacer un chequeo un poco mas complicado:
  - if(0== C[x][v]-RF[x]+SC[v])....

Restando/sumando m en tiempo $O(n)$

- Si lo estamos haciendo manualmente, hay una enorme diferencia: nuestros ojos son muy capaces de escanear rapidamente si una entrada es 0 o no, pero hacer la cuenta mental de si esa entrada menos un número mas otro número es igual a 0, demora tiempo y puede inducirnos a errores.
- Pero para una computadora eso no es problema: tanto
  - if(0== C[x][y]) como
  - if(0== C[x][y]-RF[x]+SC[y])
- demoran (casi) lo mismo, pues restar/sumar es O(1).

Restando/sumando m en tiempo $O(n)$

- Aca es donde entra lo de “suposiciones razonables”.
- Estamos suponiendo que los números involucrados estan en el rango de los números para los cuales la computadora es capaz de restar/sumar en O(1).
- Pej, en una computadora de 64 bits, si los pesos son menores o iguales que $1,8 \times 10^{19}$ entonces la suposición es cierta, y es bastante razonable suponer que los pesos serán menores o iguales que ese número:seria raro encontrar un problema en donde haya que usar pesos de mas de 64 bits.
- Pero si tenemos un problema con una matriz de pesos de 2048 bits entonces las operaciones aritméticas demorarán mas tiempo.

Restando/sumando m en tiempo $O(n)$

- En el caso razonable que podamos implementarlo porque los números no son excesivamente grandes, en realidad uno deberia:
- Calcular temp=C[x][v]-RF[x]+SC[v].
- Hacer if(0==temp)
  - Si el if se satisface, etiqueteamos la columna v, y todo lo otro que querramos hacer.
  - Si el if falla, estamos entonces en el caso que hablamos antes, que v NO ESTA en $\Gamma(S)$.
    - Por lo tanto usamos temp para actualizar mS[v] haciendo if(temp<mS[v]) mS[v]=temp;
- La mejor forma de entender todos los detalles de esta implementación es codificarlos, obviamente, pero al menos esto les dará una idea de cómo se hace.