Problema 1 - Manager Mauro

Autor: Josué Rodríguez Ramírez

Análisis del Problema:

Mauro tiene un espíritu deportivo tan grande, que paralelamente con sus estudios ha decidido también manejar el equipo de fútbol de su facultad MATCOM. Dedicar su tiempo a esto no es impedimento (cree él) para seguir obteniendo buenos resultados académicos. La verdad sea dicha, Mauro no tiene ni idea de fútbol. De hecho, varias veces se ha preguntado a sí mismo cómo llegó a esa importante posición. Lo bueno es que sí sabe de matemática y porgramación, así que decide formar un equipo (teóricamente) poderoso. Mejor aún, el equipo más (teóricamente) poderoso posible.

Se debe formar un equipo de p jugadores en distintas posiciones y k espectadores VIP (comisión de embullo). La facultad tiene n personas. Para cada persona i se conoce su valor a_i como espectador VIP y su valor p_{ij} como jugador en la posición j.

Ayude a Mauro haciendo un algoritmo que calcule el poder (valor) del mejor equipo posible. Un equipo es mejor que otro si tiene más valor.

Explicación del problema

Este es un problema de combinatoria. Probar todas las combinaciones posibles tiene una complejidad $n!$, muy ineficiente. Para resolver este problema en orden polinomial usaremos el *algoritmo Húngaro*.

Este es un algoritmo de optimización el cual resuelve problemas de asignación en tiempo $O(n^4)$
aunque existe una implementación en $O(n^3)$.

El algoritmo modela un problema de asignación como una matriz de costos n×m, donde cada elemento representa el costo de asignar el n-ésimo jugador a la m-ésima posición. Como es un problema de maximización el costo de la matriz necesita ser modificado para que la minimización de sus elementos lleve a una maximización de los valores de costo originales. Todos los elementos son restados por el valor máximo de la matriz entera.

Después de esto se siguen los siguientes pasos:

> ⓘ **Paso 1: Restar los mínimos de cada fila**

Para cada fila de la matriz se selecciona el elemento con el valor más bajo y se lo resta de cada elemento en esa fila.

ⓘ Paso 2: Restar los mínimos de cada columna:

De manera similar, se selecciona para cada columna el elemento con el valor más bajo y se lo resta de cada elemento en esa columna.

ⓘ Paso 3: Cubrir todos los ceros con un mínimo número de líneas

Se deben cubrir todos los ceros en la matriz resultante del paso 2 usando un número mínimo de líneas horizontales y verticales, ya sea por filas o columnas. (Para esto utilizaremos emparejamiento perfecto)

Si se requiere un total de n líneas para cubrir todos los ceros, siendo n igual al tamaño n por n de la matriz, se tendrá una asignación óptima entre los ceros y por tanto el algoritmo se detiene.

De lo contrario, si se requieren menos de n líneas para cubrir todos los ceros en la matriz, se continúa con el paso 4.

ⓘ Paso 4: Crear ceros adicionales:

Se selecciona el menor elemento de la matriz (llamado k) que no esté cubierto por una de las líneas realizadas en el paso 3.

Se resta el valor de k de todos los elementos que no están cubiertos por líneas. Posteriormente se suma el valor de k a todos los elementos que están cubiertos por la intersección de dos líneas.

Los elementos que están cubiertos por una sola línea se dejan tal como están. Después de realizar este paso, se regresa al paso 3.

Correctitud

Este algoritmo de asignación hace el método Gauss-Jordan, en primera instancia, tomando los menores valores de cada fila y restándola a los demás valores de la fila, y lo mismo con cada columna. Obteniendo ceros en las posiciones de los posibles valores que llevan al máximo.

La implementación se basa, entonces, en hacer un matching-perfecto en el grafo bipartito generado entre los "jugadores" y las "posiciones". En caso de que no sea bipartito, se toman los subconjuntos bipartitos (connected_components) de G.

En el libro Introduction to Algorithms de Thomas H. Cormen aparece la explicación de este algoritmo. Para ello enuncia este teorema (demostración en página 968)

Teorema: Sea G = (V, E), donde V = L ∪ R, es un gráfico bipartito completo donde cada arista (l, r) ∈ E tiene peso w(l, r). Sea h un etiquetado de vértices factible de G y Gh sea el subgrafo de igualdad de G. Si Gh contiene un emparejamiento perfecto M*, entonces M* es una solución óptima para el problema de asignación en G.

El objetivo ahora se convierte en encontrar una coincidencia perfecta en un subgrafo de igualdad. ¿Qué subgrafo de igualdad? ¡No importa! Tenemos libertad para no solo elegir un subgrafo de igualdad, sino para cambiar qué subgrafo de igualdad elegimos a medida que avanzamos. Solo necesitamos encontrar una coincidencia perfecta en algún subgrafo de igualdad.
Quiere decir que podemos tomar el grafo, luego de hacer las transformaciones, de tal manera que si encontramos un emperejamiento perfecto, también encontramos la asignación óptima.

¿Cómo creamos el grafo para el emparejamiento? Las aristas que participan son aquellas que tienen costo 0 de los jugadores a las posiciones (estas son las que nos interesan, porque nos lleva al valor máximo del equipo) Por esta razón, si el emparejamiento no es perfecto, ($|M| < max(n, m)$) debemos buscar aumentar la cantidad de ceros (esta es la idea de Camino aumentativo).

Para ello, hacemos el paso 4 descrito anteriormente y volvemos a buscar el emparejamiento hasta que sea perfecto.

¿Cómo sabemos que el algoritmo termina? Es simple, al aumentar al menos una arista más en cada proceso de la búsqueda de un empearejamiento mayor, este emparejamiento puede crecer, hasta ser perfecto, nunca será mayor que $max(n, m)$ porque si no, no sería perfecto.

Complejidad Temporal

Para calcular el emparejamiento, se usa :

```
nx.algorithms.bipartite.maximum_matching(G)
```

Este método de la biblioteca *networkx* desarrolla el algortimo Hopcroft-Karp para econtrar emparejamiento perfecto en grafos bipartitos a partir de caminos aumentativos. Su complejidad en el caso peor es $O(|E| * \sqrt{|V|})$

En el algoritmo, las primeras líneas (Pasos 1 y 2) se toma como tiempo $O(|E| * |V|)$. El bucle while siguiente itera como máximo n veces, ya que cada iteración aumenta el tamaño del emparejamiento M en 1. Cada prueba dentro del ciclo puede tomar un tiempo constante simplemente comprobando si $|M| < n$, cada actualización de M en la línea 9 toma tiempo $O(|V|)$, y las actualizaciones para la nueva matriz adjunta toman tiempo $O(|E| * |V|)$.

Como vemos, el costo total del algoritmo propuesto es $O(|E|^2 * |V|^{1,5})$ y en el caso promedio $O(|E| * |V|^{2,5})$ aunque existe una mejora de este algoritmo en $O(|V|^3)$ cambiando Hopcroft-Karp por el algoritmo mencionado en el libro "Computing a maximum cardinality matching in a bipartite graph in time $O(n^{1.5} * \sqrt{\frac{m}{logn}})$" por Alt, H.; Blum, N.; Mehlhorn, K.; Paul, M. (1991).

Bibliografía

- Thomas H. Cormen, Charles E. Leiserson, Ronald Rivest,Clifford Stein, Introduction to Algorithms, Fourth Edition.
- Alt, H.; Blum, N.; Mehlhorn, K.; Paul, M. (1991), "Computing a maximum cardinality matching in a bipartite graph in time $O(n^{1.5} * \sqrt{\frac{m}{logn}})$"