Composición de grafos k-regulares

Sea k el grado deseado, y n el número de vértices.

Entonces, k=n+1, y k\*n debe ser par y n > k +2.

Si k=2, entonces k=n-1

((k == 1) or (k == 2 and n > 2) or n-1 == k) or n > k+2 and (k\*n) % 2 == 0

Grafo de orden 2, si n > 2

G de orden 3, si n=4 o n= 6 o n=8

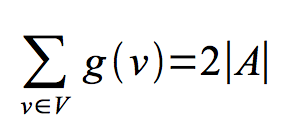
Grafo de orden 4, si n=5 o n=8 o n=12

Grafo de orden 5, n=6 o n=8

Esto de abajo es una pista, pero no es del todo exacto:

<https://math.stackexchange.com/questions/11402/existence-of-k-regular-graph>

The sum of the degrees is 2|E|. Therefore the sum of the degrees must be an even number. Since an odd times an odd is always an odd, and the sum of the degrees of an k-regular graph is k\*n, n and k cannot both be odd.



# Primera versión del algoritmo

Se ha escrito un primer algoritmo que asigna las nuevas aristas por orden de peso, de menor a mayor, pero no resuelve todos los casos (por ejemplo, con n=6 y k=4).

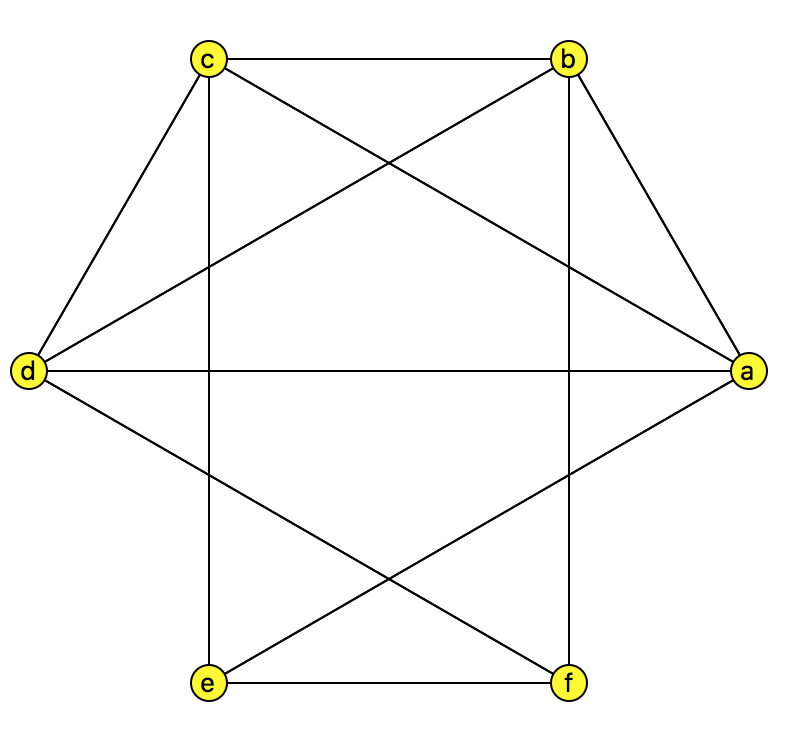
El que tendría que ser el penúltimo movimiento resulta ser el último porque el siguiente ya está asignado, entre dos vértices con orden 3.

El orden de creación de las aristas es el siguiente:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Orden | V1 | V2 | Peso V1 | Peso V2 |
| 1 | B | C | 1 | 1 |
| 2 | D | E | 1 | 1 |
| 3 | F | A | 1 | 1 |
| 4 | B | D | 2 | 2 |
| 5 | C | E | 2 | 2 |
| 6 | F | B | 2 | 3 |
| 7 | A | C | 2 | 3 |
| 8 | D | F | 3 | 3 |
| 9 | E | A | 3 | 3 |
| 10 | B | E | 4 | 4 |
| 11 | C | D | 4 | 4 |

El movimiento 11 debería haber sido C 🡪 F, seguido de D 🡪 A.

Queda un grafo parecido a éste (no exactamente el ejemplo):



… que, naturalmente, no es regular, pues los dos vértices de abajo tienen un peso inferior al resto.

# Análisis del nuevo algoritmo

El nuevo algoritmo puede seguir la regla estipulada en el actual; es decir: asignar aristas entre nodos, en orden de peso, de menor a mayor. Sin embargo, si llega a quedarse sin posibles movimientos y no ha logrado el objetivo, debería volver atrás las veces que sea necesario hasta conseguir alcanzarlo. En el caso mostrado anteriormente, tan sólo deshaciendo el último movimiento y utilizando el otro único posible como alternativa, hubiera encontrado la solución.

Evidentemente, resultaría mucho más eficiente adaptar el algoritmo de esta manera, en lugar de utilizar un algoritmo genérico, aunque es posible que un BFS pudiera cumplir aquí perfectamente su cometido, siendo el orden de los vértices equivalente a la profundidad del árbol.

Para implementar el BFS haríamos lo siguiente: en lugar de crear directamente las aristas, podría crearse una lista con el mismo contenido, como una lista de nodos explorados, alimentando al mismo tiempo la frontera, hasta encontrar la solución, o llegar a un punto en el que no puedan hacerse más movimientos. En este último caso, se recuperaría el nodo más reciente de la frontera, y se intentaría buscar un movimiento alternativo que no estuviera ya en la lista de nodos explorados, continuando así el proceso. Este algoritmo llegaría eventualmente a encontrar la solución.

Pero, ¿cómo implementar la expansión de nodos? Evidentemente, debe tenerse en cuenta el peso de cada nodo, y no devolver configuraciones previamente generadas.

Esto me lleva a considerar que un IDA\* sería el método ideal (ya que podemos recuperar estados de la frontera en orden de peso ascendente, siendo este peso el grado mayor de la configuración actual del estado).

Esto implica muchos más estados generados que con el algoritmo anterior, pero el método garantiza la resolución del problema.

## Simulación

Sea n=6 y k=4, como en el ejemplo anterior, y el conjunto de vértices es , mientras que el estado inicial (las aristas): .

|  |  |  |  |
| --- | --- | --- | --- |
| Estado | Expansión | Explorados | Frontera |
| [] |  |  | [] |
| [] |  | [] |  |
| [] | (a,b),(a,c),(a,d),(a,e),(a,f),(b,c),(b,d),(b,e),(b,f),(c,d),(c,d),(c,e),(c,f),(d,e),(d,f),(e,f) | [] | [] |
| (a,b) |  | [],(a,b) |  |
| (a,b) |  | [],(a,b) | (a,b) |
| (a,c) |  |  |  |