Punto 2 - GrafoBC

Identificación:

Juan Andrés Romero -202013449

Luccas Rojas -201923052

Explicación:

Para poder solucionar el problema de los grafos BC, partimos de reciclar el código de la tarea 5 que teníamos para hallar si un grafo era bipartito y con base en esto encontrar la diferencia mínima de estos conjuntos. IsBipartite devuelve 2 listas las cuales representan los conjuntos disyuntos de nodos del grafo separados por cada uno de los colores del grafo bipartito. Partiendo de esto, intentamos pensar el algoritmo de diferentes formas y no encontrábamos una solución para el problema que funcionara en todos los casos. Luego de analizar muy bien el problema, decidimos reducirlo a uno más sencillo, el cual constaba de sacar la longitud de ambos conjuntos de los grafos bipartitos y encontrar la posible combinación entre dichos conjuntos por cada grafo de tal modo que al final se minimizara la diferencia entre ellos. Sin embargo, esta reducción solo fue de utilidad para hacernos llegar a la reducción definitiva, la cual resultó en convertirse en el problema famoso de la diferencia mínima de sumas entre subconjuntos. Para llegar hasta dicha reducción pensamos que al fin de cuentas lo realmente importante era la diferencia entre ambos conjuntos que componían un grafo. Este número representa la diferencia entre los nodos de un color y los del otro por cada grafo. El cual aumentará o disminuirá la diferencia entre los subconjuntos una vez los grafos sean unidos. A partir de esto, corrimos la implementación de si un grafo es bipartito o no para cada grafo y sacamos la diferencia entre las longitudes de los conjuntos de cada color para cada grafo. Estas diferencias las almacenamos en una lista y posteriormente con ayuda de una solución dinámica del problema de la diferencia mínima entre la suma de subconjuntos, podemos resolver el problema original.

No obstante, esta no fue la única posible solución, al principio teníamos una solución recursiva para el algoritmo, pero la complejidad temporal era muy alta, ya que era exponencial.

```
def subconjuntoMinimo(i:int,lista:list,total1:int, total2:int)->int:
    if i<0:
        return abs(total1 - total2)
    else:
        sum=subconjuntoMinimo(i-1,lista,total1, total2+lista[i])
        res=subconjuntoMinimo(i-1,lista,total1+lista[i], total2)
        return min(sum,res)
```

Luego, para intentar encontrar otra solución intentamos un algoritmo greedy, que ordenaba la lista de enteros y comenzaba a restar al entero más grande el siguiente entero, hasta el punto en el que el resultado de la resta fuera menor que 0. En ese punto, se convertía el valor de negativo a positivo y se volvía a restar hasta que fuera 0. Este algoritmo funcionó para la gran mayoría de los casos, el problema que se presentaba era que si los 3 valores más grandes eran iguales a la suma del resto de números el algoritmo fallaba, este algoritmo era de complejidad O(n*log(n)) pero no funcionaba para todos los casos. Finalmente, la solución que escogimos utiliza programación dinámica, por lo que su complejidad es de O(N*K), donde N es el número de elementos de la lista y K es la suma de todos los números del arreglo.

Análisis de complejidad:

La complejidad temporal del algoritmo es de N*K+N*(V+E), donde N es el número de grafos, K es la suma total de las diferencias, V es el número de vértices en cada grafo y E es el número total de arcos en cada grafo. El algoritmo de solución se basa en calcular si existe o no una manera de dividir la lista de diferencias en 2, de tal forma que la diferencia de las sumas entre subconjuntos sea 0. Esto se puede resolver comparando el estado actual con la suma requerida (donde la suma requerida es la mitad de la suma total cuando es par y suma total + 1 cuando es impar) y se puede decidir entre sumarlo o no al subconjunto. Como esto utiliza estados actuales y anteriores se puede transformar en una solución de programación dinámica basada en los valores cambiantes, los cuales son el tamaño del arreglo y la suma requerida. La matriz de estados en este caso guarda booleanos, los cuales indican si para cada elemento anterior al índice N se puede generar subconjuntos cuya suma sea el valor indicado en las columnas K. Si sí se puede dar dicho valor, el valor de la casilla sería true, si no sería false. A partir de esto, la matriz se crea en un tiempo de N*K y es llenada en un tiempo también de N*K.

En general, el algoritmo se basa en realizar el proceso de sacar los dos subconjuntos de un grafo, que se genera por el algoritmo isBipartite, cuya complejidad es de O(V+E) por el BFS. Este proceso de sacar los subconjuntos se realiza N (número de grafos) veces. Luego, se almacena la diferencia de estos subconjuntos en una lista y se encuentra la matriz de resultados del problema descrito anteriormente, cuya complejidad es de 2N*K. Finalmente, se recorre la última fila de la matriz para dar con la solución, dentro de esta se revisa el elemento con valor de True en el cual el valor de la columna de la suma sea más cercano a 0. Una vez encontrado, el valor de la respuesta es la suma total – 2*el valor de la columna.

Del mismo modo, la complejidad espacial del algoritmo es de N*K+N+N*(V+E), debido a que se tiene que almacenar la matriz en memoria, la lista de diferencias entre los subconjuntos de un grafo, las listas de adyacencias de cada grafo y varias constantes externas como contadores y retornos.

Comentarios:

Finalmente, la solución es óptima en el sentido que la complejidad tanto espacial como temporal son polinomiales y de grado 2, acotadas como O(n^2). No obstante, basta comentar que la complejidad espacial posiblemente podría ser reducida a una complejidad lineal eliminando la matriz de estados y solo dejando la fila anterior para poder resolver para la fila actual, sin embargo, no logramos encontrar la forma de reducir el algoritmo hasta dicho punto. Un punto para recalcar es que el algoritmo ahorra bastante espacio en memoria y tiempo al no trabajar directamente con los grafos, sino con una representación numérica de estos.