1 Ejercicio 3

a) Explicación de la heuristica

La idea básicamente es, dado un grafo G, ir tomando pares de nodos de manera tal que, dado un nodo *nodoActual*, ya perteneciente a la partición parcial (de ahora en más P) que estamos construyendo, tomamos (si es posible) un nodo u adyacente a *nodoActual* y un nodo v adyacente a u. Tanto u como v no deben haber sido agregados todavia a P. El nodo u puede ser cualquier adyacente a *nodoActual*, pero v debe ser un nodo adyacente a u tal que el peso de la arista (u, v) sea máximo sobre los pesos de las demás aristas incidentes a u (aristas que unen a u con nodos no pertenecientes a P). Una vez hecho esto, tratamos de ubicar a u y a v de todas las maneras posibles en P, agregandolos finalmente en las ubicaciones que minimicen la suma total de los pesos intrapartición (esto representa la mejor manera que podemos agregar los nodos). Finalmente, *nodoActual* pasará a valer v y repetimos lo explicado para este nuevo valor. Si pudimos tomar al nodo u pero no existe v que cumpla con las condiciones pedidas ($(u, v) \in E(G), v \notin P$), entonces solo ubicamos a u de la mejor manera. Más abajo diremos que sucede cuando nos encontramos en el caso que ni siquiera fue posible encontrar a u (porque *nodoActual* no tiene adyacentes o todos sus adyacentes ya pertenecen a P).
Inicialmente ($P = \{\emptyset_1, \emptyset_2, ..., \emptyset_k\}$) el algoritmo va a tomar (si esto es posible) el par de nodos adyacentes donde el peso de la arista que los une es máximo sobre todas las demás aristas del grafo. En principio, no hay un criterio de elección para cuando existen varios pares de nodos que cumplen esto, por lo cual la elección será arbitraria. Si no existe ningún par de nodos que cumpla esto, entonces no existe ningún par de nodos adyacentes, por lo cual cualquier partición que elijamos va a ser solución. Particularmente el algoritmo generará una partición donde ubique a todos los nodos en un mismo conjunto. En el caso de que sí exista este par de nodos, se procederá a ubicarlos dentro de P en conjuntos distintos (porque queremos ubicarlos de la mejor manera posible). Una vez hecho esto, si el par de nodos agregados fue u y v, el paso siguiente será guardar todos los nodos adyacentes a v en alguna estructura de datos C y elegir como *nodoActual* a u, comenzando así el proceso descripto en el párrafo anterior. Podemos decir ahora, que cuando nos encontramos en el caso en el que *nodoActual* no tiene adyacentes, recurriremos a tomar un nodo de C.
Como en C podría no haber ningún nodo (o tener nodos que ya agregamos a P), tendríamos que fijarnos si todavía queda algún nodo por agregar a la partición (ya que en C van a estar los nodos pertenecientes a la componente conexa que estamos recorriendo). Entonces en el caso de que todavía exista algún nodo que no agregamos a la partición y C sea vacía, el algoritmo tomará un nodo u de los que faltan (en particular, el de número menor). Si no puede tomar a u junto con otro nodo adyacente a él, v (al igual que antes, porque no vale $(u, v) \in E(G), v \notin P$) tratará de ubicar a u de la mejor manera en P y comenzará el proceso descripto anteriormente con $nodoActual = u$. De lo contrario, lo tomará junto con v, ubicará a los dos nodos en P de la mejor manera y comenzará de nuevo el proceso con $nodoActual = v$. Si C sí tenía algún nodo que no agregamos P, el procedimiento es similar: el algoritmo toma el nodo $u \in C$, busca un adyacente a él (llamemoslo v, al igual que antes, $v \notin P$), se asegura que valga que la arista que los une tenga peso máximo y los ubica en la partición, repite el proceso con $nodoActual = v$. Si no pudo encontrar a un v que cumpla lo pedido, ubica solo a u en P y repite el procedimiento con $nodoActual = u$.

Para reforzar la idea, dejamos a continuación un pseudocódigo de alto nivel.

1: **procedure** HGolosa($G(V, E)$, $w : E \to float$)
2: Si $E == \emptyset$ devolver $P = \{V, \emptyset_1, ..., \emptyset_k\}$
3: Si $k == 1$ devolver $P = \{V\}$

4: $C = \emptyset, P = \emptyset$
5: Tomar $u_m, v_m \in V$ tales que $(u_m, v_m) \in E \wedge w((u_m, v_m)) \leq w((u, v))\ \forall u, v \in V, (u, v) \in E$
6: Insertar a u_m y v_m en conjuntos distintos pertenecientes a P
7: $nodoActual = u_m$
8: Agregar en C todos los adyacentes a v_m

9: **while** $\exists\ u \in V \wedge u \notin p, p \subset P$ **do**
10: $ady \leftarrow -1$
11: **if** $nodoActual \neq -1 \wedge \exists\ u \in V, u \notin p \subset P, (u, nodoActual) \in E$ **then**

12: $ady \leftarrow u$
13: Agregar en C todos los adyacentes a *nodoActual* que no pertenezcan a $p \subset P$
14: **else**
15: **if** $\exists\ u \in C \land u \notin p, p \subset P$ **then**
16: $ady \leftarrow u$
17: **else**
18: Tomo algún ady tal que $ady \in V, ady \notin p \subset P$
19: **end if**
20: **end if**

21: **if** $\exists\ v \in V \land v \notin p \land (ady, v) \in E, p \subset P$ **then**
22: Agregar *par* y *ady* a P de la mejor manera posible
23: $nodoActual \leftarrow par$
24: Agregar a C, todos los adyacentes de *ady* que no pertenezcan a P
25: **else**
26: Agregar *ady* a P de la mejor manera posible
27: **if** $C \neq \emptyset$ **then**
28: $nodoActual \leftarrow m, m \in C$
29: **else**
30: $nodoActual \leftarrow -1$
31: **end if**
32: **end if**
33: **end while**
34: **end procedure**

b) Calculo del orden de complejidad

Podemos ver que, en cada iteración el algoritmo agrega a la partición parcial P, a lo sumo dos vértices, con lo cual la cantidad total de iteraciones es lineal a la cantidad de nodos, o sea n. Para analizar el costo de una iteración, podemos comenzar por ver que, como mínimo, siempre vamos a hacer también una cantidad lineal a n de operaciones. Esto es debido a que el if de la linea 21 siempre se va a ejecutar y, como lo que hace es fijarse si existe algún nodo adyacente a *ady*, no agregado a P, no le queda otra que fijarse nodo por nodo a ver si alguno cumple esto. Para entender lo anterior es importante agregar que, en la implementación, trabajamos con matrices de adyacencia y tenemos un arreglo de booleanos, que nos permite saber si un nodo fue agregado a P o no. Por ahora vimos que una iteración es $\Omega(n)$, a esto tenemos que sumarle el costo de agregar el o los nodos a P. Es fácil ver que el peor caso es cuando hay que agregar dos nodos a P de la mejor manera, y, el hecho de asumir que en un peor caso siempre se agregan dos nodos no contradice el cáculo que venimos haciendo hasta ahora, ya que la cantidad de iteraciones seguiría siendo lineal y el if de la linea 21 es independiente de este hecho.