



Análisis y Diseño de Algoritmos

Parcial 4 (Noviembre 2025)

Nombre:

Grupo:

1.(10 puntos) Parte teórica.

Dado un grafo no dirigido $G(V, E)$ que no contiene lazos ($\forall i \in V : (i, i) \notin E$) se desea encontrar un conjunto $S \subseteq V$ de **tamaño mínimo** tal que cada arista $(u, v) \in E$ tiene al menos uno de sus extremos en S .

- (8 pts.) Diseñar el mejor algoritmo voraz posible para resolver el problema. Se debe indicar claramente la estructura de la solución, el estado inicial de la misma, la función de terminación, la función objetivo, los candidatos factibles en cada paso y la función de selección.
- (2 pts.) Demostrar la corrección del algoritmo. Es decir, si encuentra la solución óptima en todo caso o no.

Estructura de la solución: Un conjunto de nodos S . Inicialmente vacío.

Función objetivo: el cardinal del conjunto, $|S|$. Buscamos minimizarlo.

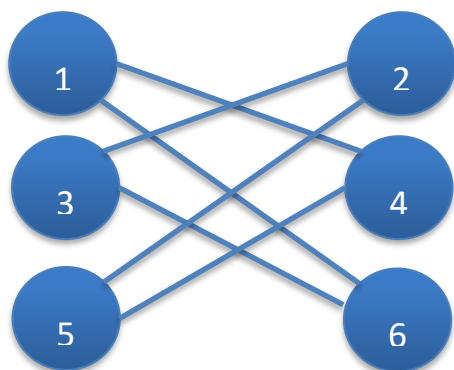
Función de terminación: Todas las aristas deben estar cubiertas. $\forall (u, v) \in E : u \in S \vee v \in S$

Candidatos factibles: Vértices del grafo que aún no han sido seleccionados.

$$Cand = \{c \mid c \in V - S\}$$

Función de selección: Seleccionaremos el vértice sobre el que incidan más aristas (el de mayor grado).

El algoritmo voraz no encuentra la solución en todo caso. Para el grafo indicado la solución dada por el voraz sería $S=\{1,2,3,4\}$ cuando la solución óptima es $\{1,3,5\}$ o $\{2,4,6\}$



2. (10 puntos) Parte práctica.



Implementa el algoritmo voraz diseñado en el ejercicio anterior.

```
public Set<Integer> getConjuntoCobertura() {
    Set<Integer> res = new HashSet<Integer>();
    Grafo aux = new Grafo(grafo);

    while(aux.numAristas() > 0) {
        //Escogemos el vértice de mayor grado de los no incluidos
        Integer nodo= nodoMayorGrado(aux);
        res.add(nodo);
        // eliminamos las aristas en las que participa el nodo
        eliminarAristas(aux, nodo);
    }

    return res;
}

private void eliminarAristas(Grafo g, Integer i) {
    Set<Integer> conectados = new HashSet<> (g.sucesores(i));
    for (Integer nodo : conectados) {
        g.removeArista(i, nodo);
    }
}

/*Encuentra el nodo de mayor grado que no está ya en la solución
*Realmente no hace falta comprobar que no esté en la solución, porque al ir
eliminando las aristas de los
* nodos que vamos incluyendo, nunca se seleccionarán en pasos posteriores (su
grado es 0).
* Si el grado de todos los vértices es 0, nunca se invocará este método.
*/
private Integer nodoMayorGrado(Grafo g) {
    Integer nodoMayor = null;
    Integer mejorGrado = -1;
    for (Integer nodo : g.nodosConectados()) {
        Integer grado = g.grado(nodo);
        if (nodoMayor == null || grado > mejorGrado) {
            nodoMayor = nodo;
            mejorGrado = grado;
        }
    }
    return nodoMayor;
}
```