```
void Solucion::Evaluar(Problema p) {

	// calcular los grados de los nodos

	int * grados;                 → O(1)
	int nodo_seleccionado;        → O(1)
	bool solucion = false;        → O(1)


	grados = new int[p.N] ;       → O(1)

	for(int i = 0 ; i < p.N ; i++) {     //Solo para calcular grados
		for(int j = i ; j < p.N) {
			if(p.adyacencia[i][j] == 1){
				grados[i]++ ;
				grados[j]++ ;
			}
		}
	}

	while(recubrimiento.size() < p.N and !solucion) {          → for(i=0; i < p.N && !solucion;
i++)          → \sum^{(n-1)} n    = O(n^2)
		nodo_seleccionado = getMaximo(grados, p.N);   → O(n)
		solucion = añadirNodo(nodo_seleccionado);     → O(n)

	}

}


bool Solucion::añadirNodo(int nodo, int * grados) {              → O(n)
	bool solucion = true;                          → O(1)
	grados[nodo] = 0;                              → O(1)
	for(int i=0; i<p.N; i++) {                                   → O(n)
		if(adyacencia[nodo][i] == 1) {         → max(O(1),0) = O(1)
			grados[i]--;                   → O(1)
			adyacencia[nodo][i] = 0;       → O(1)
		}

	}
	recubrimiento.push_back(nodo);                 → O(1)

	for(int i=0; i<p.N && !solucion; i++) {  //Solo para comprobar si ha llegado a la solución
		if(grados[i] > 0) {
			solucion  = false;
		}
	}

	return solucion;                               → O(1)
}
```

```
int getMaximo(int [] vector, int tam){                    → O(n)
      int max = 0 ;                          → O(1)
      for(int i = 1 ; i < tam ; i++){              → O(n)
            if(vector[i] > vector[i-1])     → max(O(1),0) = O(1)
                  max = i ;                → O(1)
      }
      return max ;                           → O(1)
}
```