Ejercicio 9

Rojo Mata Daniel

Complejidad en tiempo y espacio

- **sonVecinos**

```
1 public static boolean sonVecinos(String identificador1,
2                                   String identificador2, Grafica grafica) {
3     return grafica.buscarArista(identificador1, identificador2);
4 }
```

Se hace uso del método `buscarArista` para revisar si dos vértices son vecinos. Es por ello que el método `sonVecinos` tiene la misma complejidad que el primer método. El cálculo de la complejidad del método `buscarArista` se hace en la sección de Anexos, 1. Heredando lo anteriormente mencionado, se concluye que:

$$T(m) = 8m + 1$$

$$M(n) = 0$$

Esto es, $T(n) \in O(n)$ y $M(n) \in O(1)$.

- **resgresaMatrizAdyacencias**

```
1 public static int[][] regresaMatrizAdyacencias(Grafica grafica) {
2     LinkedList<Grafica.Vertice> vertices = grafica.darVertices();
3     int nVertices = vertices.size();
4     int[][] matriz = new int[nVertices][nVertices];
5
```

```
6       for (int i = 0; i < nVertices; i++) {
7           for (int j = 0; j < nVertices; j++) {
8               if (sonVecinos(vertices.get(i).darIdentificador(),
9                   vertices.get(j).darIdentificador(),
10                  grafica)) {
11                  matriz[i][j] = 1;
12              } else {
13                  matriz[i][j] = 0;
14              }
15          }
16      }
17      return matriz;
18  }
```

Dentro del bloque `if-else 8-14` se toma el peor de los casos en el cual se tenga un mayor número de operaciones elementales, en esta situación, se contempla el bloque `if 8-12`.

Se usa el método `sonVecinos` en la línea 8, por lo que se utiliza su función T En la línea 11 se está haciendo una asignación y se ha accedido a un elemento del arreglo, por lo que se cuentan 2 operaciones elementales. Así, en este bloque se tiene:

$$T_1(n) = T_{sonVecinos}(n) + 1$$

Dentro del ciclo `for 7-15`, la función T_1 se aplica, a lo sumo, n veces, sin embargo, se agregan las contribuciones de las operaciones para inicializar el ciclo (`j = 0; j<nVertices; j++`). Éstas últimas contribuyen a 4 operaciones elementales.

Es así que se tendría: $n\big(T_1 + 4\big)$

Ahora, lo anterior se aplica a lo sumo n veces debido al ciclo `for 6-16`, añadiendo las 4 operaciones para inicializar el ciclo y lo anterior se tiene:

$$T(n) = n\Big(n\big(T_1 + 4\big)\Big) + 4$$

Simplificando:

$$T(n) = n^2 T_1 + 4n^2 + 4$$

Sustituyendo el valor de T_1 se tiene:

$$T(n) = n^2\big(T_{sonVecinos}(n) + 1\big) + 4n^2 + 4$$

Sustituyendo $T_{vecinos}$:

$$T(n) = n^2\big(8n + 1 + 1\big) + 4n^2 + 4$$

$$T(n) = 8n^3 + 2n^2 + 4n^2 + 4$$

$$T(n) = 8n^3 + 6n^2 + 4$$

Por otra parte, se crea una matriz de tamaño n^2 en la línea 4, por lo que se consume memoria extra. Es así que:

$$M(n) = n^2$$

Finalmente, $T(n) \in O(n^3)$ y $M(n) \in O(n^2)$.

1. Anexos

- **buscarArista**

```
1  public boolean buscarArista(String u, String v) {
2          boolean encontrada = false;
3          for (Arista e : this.aristas) {
4              if ((e.darU().equals(u) && e.darV().equals(v)) ||
5                  (e.darU().equals(v) && e.darV().equals(u))) {
6                  encontrada = true;
```

```
7                   break;
8               }
9           }
10          return encontrada;
11      }
```

Acceder a un atributo no toma tiempo de ejecución, es por ello que en el bloque `if 4-7` se contabilizan solo las operacoones `equals` y &&. La primera ocurre 4 veces mientras que la segunda 2, además se utiliza || en una ocasión. Finalmente, existe una asignación en la línea 6 contando un total de 8 operaciones elementales. Dichas operaciones se realizan a lo más $|E|$ veces, pues se está iterando en un ciclo de la forma `for (Arista e : this.aristas)`. Así pues, si $m = |E|$, entonces:

$$T(m) = 8m + 1$$

Esto sugiere que la complejidad del algoritmo es lineal, esto es; $T(m) \in O(m)$.

Espacialmente no se utiliza memoria extra, por lo que se asegura que:

$$M(n) = 0$$

Luego, $M(n) \in O(1)$

Gráfica dibujada

Se muestra la representación visual de la gráfica realizada en la clase `PruebaMatriz`.

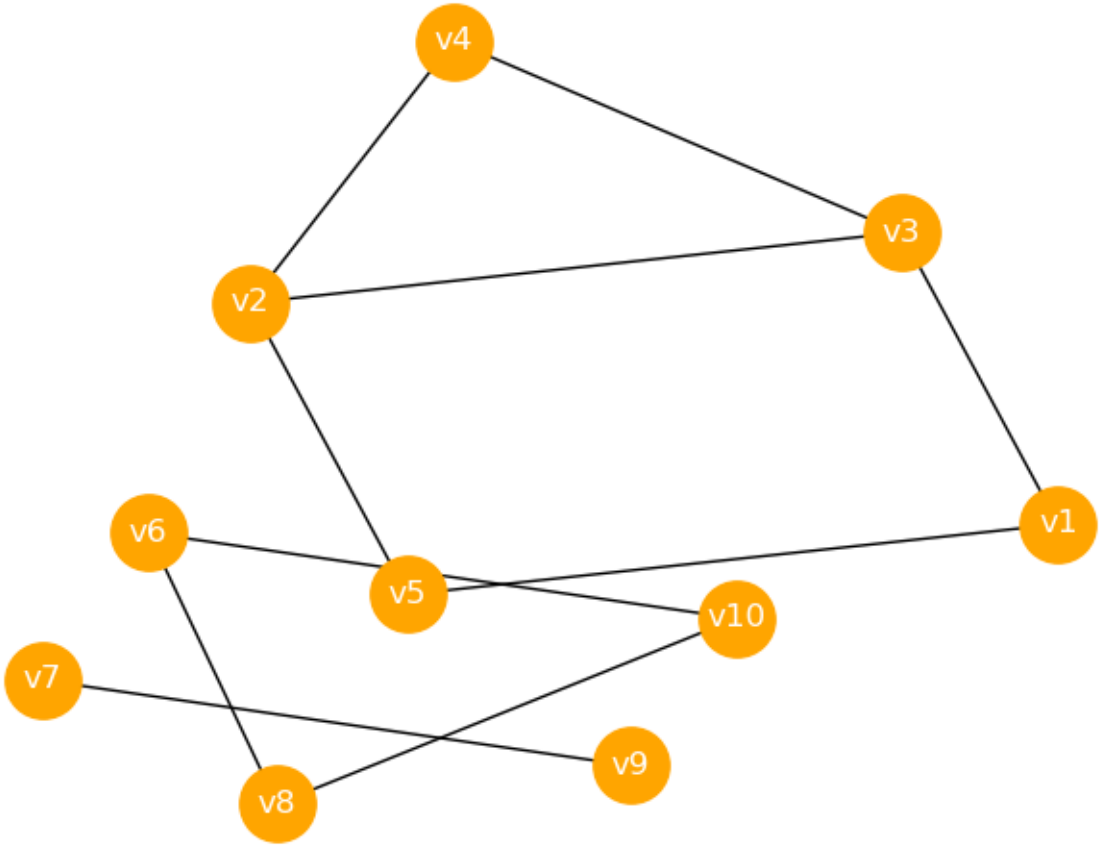

Figura 1: Gráfica con 10 vértices