1. Juan Camilo Ruiz y Sergio Guzmán
2. ALGORITMO DE SOLUCIÓN
   1. CONSTRUCCIÓN DEL GRAFO:

Para este grafo se utilizaron las siguientes funciones matemáticas de verificación del programa:

-

-

-

-

//Array for the boolean bipartition

    public int[] bipartition;

    /\*\*

     \* Graph constructor that uses a given line with the structure (v e x1 y1 x1 y2... xk yk) to create a graph bipartition.<br>

     \* @param line Given line with the forma described above

     \*/

    public Graph(String line)

    {

        //Q: The graph is bipartite

        //Data split O(n)

        String[] data=line.split(" ");

        //Number of vertexes

        int v=Integer.parseInt(data[0]);

        //Adjacency matrix

        boolean[][] adj= new boolean[v][v];

        //P1:2<=i<data.length ^ AdjPairs(data,i,adj)

        //t1:n-i

        for(int i=2;i<data.length;i+=2)

        {

            adj[Integer.parseInt(data[i])][Integer.parseInt(data[i+1])]=true;

            adj[Integer.parseInt(data[i+1])][Integer.parseInt(data[i])]=true;

        }

        //R1: AdjPairs(data,data.length,adj)

        //Marked array for each pair until data.length there is an edge in adj

        boolean[] marked=new boolean[v];

        bipartition= new int[2];

        //Regular queue with O(1) operations

        Queue<Integer> agenda=new LinkedList<>();

        int source=Integer.parseInt(data[2]);

        //Array that stores the value of the vertixes partition, which is either 0 or 1.

        int[] value=new int[v];

        //Adds one element to the first number of the bipartition

        bipartition[0]=1;

        marked[source]=true;

        agenda.add(source);

        int color=0;

        value[source]=0;

        //Cycle with complexity O(V)

        //P2:!agenda.isEmpty() ^ bip(marked,value,0)=bipartition[0]^bip(marked,value,1)=bipartition[1]

        //t2:agenda.size()

        while(!agenda.isEmpty())

        {

            //New source of bfs

            source=agenda.poll();

            //Changes the value of the given vertex to the other possible bipartition for the children found in the bfs

            color=(value[source]+1)%2;

            //Cycle that goes through all the adjoint vertexes. O(V)

            //P3:0<=i<v ^ markedValue(adj,source,i,marked,value)

            //t3:v-i

            for(int i=0;i<v;i++)

            {

                if(adj[source][i] && !marked[i])

                {

                    bipartition[color]++;

                    marked[i]=true;

                    agenda.add(i);

                    value[i]=color;

                }

            }

            //R3:markedValue(adj,source,v,marked,value)

        }

        //R2: agenda.isEmpty() ^ bip(marked,value,0)=bipartition[0]^bip(marked,value,1)=bipartition[1] ^ generalMarked(adj,v,marked,value)

    }

Al final de la construcción del grafo se guardan dos números enteros en el arreglo *bipartition* que constituyen la cantidad de elementos que hay en cada bipartición del grafo.

* 1. ALGORITMO:

1. EXPLICACIÓN DE COMPLEJIDAD:
   1. Para lo referente a la construcción del grafo la operación más costosa que se tiene es un recorrido doble para realizar el BFS (que posee una complejidad máxima de O(V^2) donde V es el número de vértices del grafo) y a nivel espacial se tiene un espacio constante de tamaño V^2 en el peor caso (matriz de adjuntas de vértices).