

# Manual de Usuario

## PROYECTO 1

Relaciones y sus propiedades

Mauricio G. Coello | A01328258 | 6 de junio del 2016

## Introducción

Una relación es un vínculo o una correspondencia. En el caso de la relación en las matemáticas, se trata de la correspondencia que existe entre dos conjuntos: a cada elemento del primer conjunto le corresponde al menos un elemento del segundo conjunto.

## Manual de Usuario

En este caso, se utilizó C++ como lenguaje para desarrollar esta utilidad, por lo que para su ejecución se requiere de un compilador tipo MinGW.

Para generar el ejecutable del código, abra la terminal de su ordenador, diríjase al directorio donde tiene el archivo .cpp. Una vez ahí, para compilar ejecute la siguiente línea de comando:

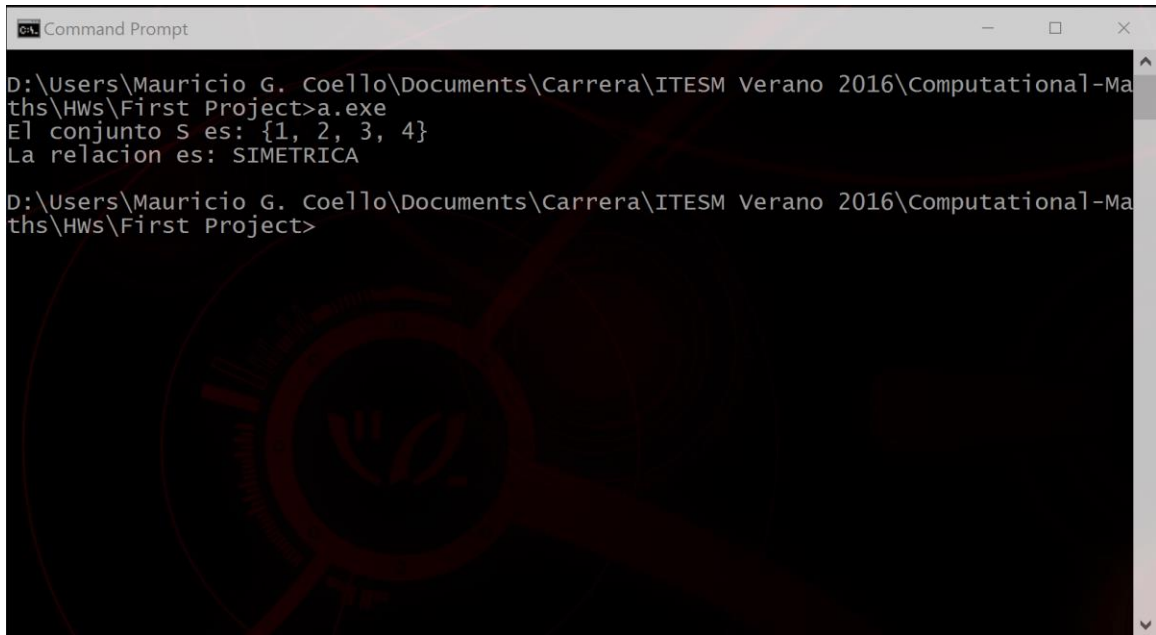
```
g++ -std=c++11 "Nombre_del_archivo".cpp
```

Para ejecutar el programa, es necesario tener un archivo .txt con el nombre de archivo "relations.txt" con el siguiente formato:

```
5
1 2
2 1
3 4
4 3
1 1
```

Siendo el primer número el que indica el número de conjuntos de la relación y que el programa va a leer.

## Ejemplo de funcionalidad



```
Command Prompt
D:\Users\Mauricio G. Coello\Documents\Carrera\ITESM Verano 2016\Computational-Ma
ths\Hws\First Project>a.exe
El conjunto S es: {1, 2, 3, 4}
La relacion es: SIMETRICA

D:\Users\Mauricio G. Coello\Documents\Carrera\ITESM Verano 2016\Computational-Ma
ths\Hws\First Project>
```

## Descripción técnica

La implementación en código de las propiedades de las relaciones se divide en 3 subfunciones que verifican si la relación cumple o no cumple con las propiedades.

Como primer paso se lee el archivo y las parejas se guardan en un vector, después se procede a detectar los miembros de la relación y guardarlos en un vector como se muestra en el siguiente código.

```
//detects all the members of the relation
for (int j=0; j<pairs.size(); j++)
{
    tmp=pairs[j];
    unique=true;
    for(int k=0; k<members.size(); k++)
    {
        if (tmp==members[k])
        {
            k=members.size();
            unique=unique*false;
        }
        else
            unique=unique*true;
    }
    if (unique)
        members.push_back(tmp);
}
```

## Reflexiva

```
bool Reflexive (std::vector<int> pairs, std::vector<int> members)
{
    //Reflexive test
    bool Reflexive = true;
    int tmp, ltmp;
    for (int i=0; i<members.size();i++)
    {
        tmp=members[i];
        //std::cout<<tmp<<std::endl;
        for (int j=0;j<pairs.size(); j++)
        {
            if(tmp==pairs[j] && tmp==pairs[j+1])
                Reflexive=Reflexive*true;
            else
            {
                if (j==(pairs.size()-2))
                    Reflexive=Reflexive*false;
            }
            //std::cout<<pairs[j]<<pairs[j+1]<<std::endl;
            j++;
        }
    }
    return Reflexive;
}
```

En este caso se utilizan 2 fors anidados para recorrer el vector que contiene los miembros y ver si en el vector parejas hay una pareja con el mismo miembro dos veces.

## Simétrica

```
bool Symmetric (std::vector<int> pairs, std::vector<int> members)
{
    //Symmetric test
    bool Symmetric = true;
    int fpair, spair;
    for (int i=0;i<pairs.size();i++)
    {
        fpair=pairs[i];
        spair=pairs[i+1];
        //std::cout<<"Looking for pair "<<spair<<", "<<fpair<<std::endl;
        for (int j = 0; j < pairs.size(); j++)
        {
            if(spair==pairs[j] && fpair==pairs[j+1])
            {
                Symmetric=Symmetric*true;
                //std::cout<<"FOUND!"<<std::endl;
                j=pairs.size();
            }
            else
            {
                if(j==(pairs.size()-2))
                {
                    Symmetric=Symmetric*false;
                    //std::cout<<"NOT FOUND!"<<std::endl;
                }
            }
            j++;
        }
        i++;
    }
    return Symmetric;
}
```

En este caso igualmente se necesitan 2 fors para recorrer el vector de parejas simultáneamente para obtener en cada iteración del primer for cada pareja y en el segundo, buscar la pareja previamente encontrada invertida.

## Transitiva

```

bool Transitive (std::vector<int> pairs, std::vector<int> members)
{
    //Transitive test
    bool Transitive = true;
    int fpair, spair;
    for (int i=0; i<pairs.size()-1;i++)
    {
        fpair=pairs[i];
        spair=pairs[i+1];
        //std::cout<<"Pair "<<fpair<<"", "<<spair<<std::endl;
        for (int j=0; j<pairs.size()-1;j++)
        {
            if(spair==pairs[j])
            {
                spair=pairs[j+1];
                //std::cout<<"looking for pair "<<fpair<<"", "<<spair<<std::endl;
                for (int k=0;k<pairs.size(); k++)
                {
                    if(fpair==pairs[k] && spair==pairs[k+1])
                    {
                        Transitive=Transitive*true;
                        //std::cout<<"FOUND!"<<std::endl;
                        j=pairs.size();
                        i++;
                        i++;
                    }
                    else
                    {
                        if (k==(pairs.size()-2))
                        {
                            Transitive=Transitive*false;
                            //std::cout<<"NOT FOUND!"<<std::endl;
                        }
                        k++;
                    }
                }
            }
            j++;
        }
        i++;
    }
    return Transitive;
}

```

En este caso utilicé 3 fors anidados, el primero es para encontrar cada pareja, el segundo compara si en alguna pareja, el segundo miembro de la pareja encontrada en el primer for, es igual al primer miembro de la pareja encontrada en el segundo for, si es así se procede al tercero, que busca si hay una pareja que contenga como primer miembro el mismo primer miembro de la pareja encontrada en el primer for y que el segundo miembro sea igual al segundo miembro de la pareja encontrada en el segundo for.

## Bibliografía

- Presentaciones de clase, Dr. Víctor de la Cueva, 2016