Universidad Católica San Pablo

PROGRAMA PROFESIONAL

Ciencias de la Computación

TÍTULO DEL TRABAJO

Tarea – Circuito Hamiltoniano

CURSO

Análisis y Diseño de Algoritmos

ALUMNOS

- Angel Josue Loayza Huarachi

SEMESTRE: V

AÑO: 2022

1. Link a GitHub:

https://github.com/angel452/ADA

2. Archivos:
    a. Source.cpp

Contiene el código main

```
#include <iostream>
#include <fstream>

#include <time.h>
#include <stdlib.h>

#include "GeneralFunctions.hpp"

using namespace std;

int main()
{
    bool** A = initializeMatrix();

    //------------------ lectura de matrizA.txt -------------------------
    string filename("matrizA.txt");
    bool number;

    ifstream input_file(filename);
    if (!input_file.is_open()) {
        cerr << "Could not open the file - '"
            << filename << "'" << endl;
        return EXIT_FAILURE;
    }

    while (input_file >> number) {
        for (int i = 0; i < V; i++)
        {
            for (int j = 0; j < V; j++)
            {
                A[i][j] = number;
                input_file >> number;
            }
        }
    }

    input_file.close();
    //--------------------------------------------------------------------

    // Pasamos la informacion a otra matriz B[V][V]
    bool B[V][V];

    for (int i = 0; i < V; i++)
    {
        for (int j = 0; j < V; j++)
        {
            B[i][j] = A[i][j];
        }
    }

    cout << "Matrix B:" << endl;
```

```
    printMatrix2(B, V);

    // ---------------------- CIRCUITO HAMILTONIANO -------------------------
    /*
    EJEMPLO DE MATRIZ
          a(1)   b(2)   c(3)   d(5)   e(6)   f(7)   g(8)
    a(1)   0      1       1      0      1      0      0
    b(2)   1                     1             1      1
    c(3)   1      1                     1             1
    d(5)          1                                   1
    e(6)   1              1                    1
    f(7)          1                     1
    g(8)                  1      1
    */

    hamilton(B);
    cout << "Fin del programa" << endl;
//-------------------------------------------------------------

}
```

b. General Functions.hpp

Contiene funciones adicionales y la función “Hamilton”

```
#include <iostream>
#define V 4
#pragma once

using namespace std;

bool** initializeMatrix()
{
    bool** temp = new bool* [V];
    for (int i = 0; i < V; i++)
        temp[i] = new bool[V];
    return temp;
}

void printMatrix2(bool M[V][V], int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            std::cout << M[i][j] << " ";
        std::cout << std::endl;
    }
    std::cout << std::endl;
}

bool isGrade2(int** A, int n, int fila) //sin usar
{
    bool flag = false;
    int grade = 0;

    for (int i = 0; i < n; i++)
    {
        if (A[fila-1][i] == 1)
        {
            grade++;
        }
    }
    if (grade == 2)
    {
```

```
        return true;
    }
    else
    {
        return false;
    }
}

void resCamino(int path[])
{
    cout << "Si hay solucion" << endl;
    cout << "Camino: " << endl;
    for (int i = 0; i < V; i++)
        cout << path[i] << " ";

    // Incluimos en la trayectoria el primer vertice
    cout << path[0] << " ";
    cout << endl;
}

bool verificador(int v, bool A[V][V], int camino[], int pos)
{
    //verificar si el vertice esta unido al anterior
    if ( A[ camino[pos - 1] ][v] == 0)
    {
        return false;
    }

    //verificar si el vertice ya se encuentra en el "camino"
    for (int i = 0; i < pos; i++)
    {
        if (camino[i] == v)
        {
            return false;
        }
    }
    return true;
}

bool hamiltonAux( bool A[V][V], int camino[], int pos) // "pos actua como un
puntero"
{
    if (pos == V) // Aca se detiene la recursividad (Caso base). Cuando
todos los vertices estan en el "camino"
    {
        if ( A[camino[pos - 1] ][ camino[0] ] == 1) //si el ultimo vertice
conecta al primero... Si hay Circuito H.
        {
            return true;
        }

        else
            return false;
    }

    //recursividad
    for (int v = 1; v < V; v++)
    {
        if (verificador(v, A, camino, pos)) //verificar si es posible poner
al vertice en el "camino"
        {
            camino[pos] = v; //insertamos el primero
```

```
            if ( hamiltonAux(A, camino, pos + 1 ) == true) //pos+1 es el
siguiente vertice
            {
                return true;
            }
            camino[pos] = -1; // es el caso cuando no nos lleva al primer
vertice. Descartamos al vertice
        }
    }

    return false;
}

bool hamilton(bool A[V][V])
{
    //creamos "camino"
    int* camino = new int[V];
    for (int i = 0; i < V; i++)
        camino[i] = -1;

    camino[0] = 0; //incluimos en el "camino" el primer vertice

    if (hamiltonAux(A, camino, 1) == false)
    {
        cout << "No hay solucion";
        return false;
    }

    resCamino(camino);
    return true;
}
```