Comparación de Programa. Grafo a árbol en anchura.

Para esta actividad hice una comparación entre el programa que yo realicé y uno generado por Chat GPT, ambos con el propósito de convertir un grafo en arbol en anchura.

Principales diferencias:

1. **Interfaz de usuario:** El programa que yo realicé utiliza una interfaz de usuario con colores y menús para que el usuario ingrese y visualice la información del grafo. El programa generado por Chat GPT se centra únicamente en la conversión del grafo en un árbol en anchura y no tiene una interfaz de usuario con colores ni menús.
2. **Entrada de datos:** El programa que yo realicé, el usuario ingresa los nodos y las conexiones del grafo a través de la consola en tiempo de ejecución. El programa generado por Chat GPT, los nodos y conexiones del grafo se definen directamente en el código.
3. **Dependencias:** El programa que yo realicé utiliza la función **SetConsoleTextAttribute** de la biblioteca Windows.h para manipular los colores de la consola. El programa generado por Chat GPT no tiene esta dependencia y simplemente utiliza la salida estándar.
4. **Tamaño del grafo:** El programa generado por Chat GPT utiliza un grafo fijo con 7 nodos para demostrar el algoritmo BFS. Tu programa permite al usuario especificar el tamaño del grafo y agregar nodos y conexiones según sea necesario.

En resumen, ambos programas tienen el propósito de realizar un recorrido en anchura en un grafo, pero tienen diferentes enfoques y características. El programa que yo realicé está diseñado para ser interactivo y permite al usuario ingresar datos del grafo en tiempo de ejecución, mientras que El programa generado por Chat GPT es una implementación más simple del algoritmo BFS en un grafo fijo definido en el código. La elección entre los dos programas dependerá de tus necesidades específicas y de si deseas una interfaz de usuario interactiva o una implementación más generalizada del algoritmo.

Diferencias de algoritmo de conversión:

1. Ambos programas utilizan una cola (queue) para mantener un seguimiento de los nodos que deben visitarse en orden de anchura.
2. Comienzan desde un nodo raíz dado y exploran todos los nodos alcanzables desde ese nodo antes de pasar a los nodos en el siguiente nivel de profundidad.
3. Mantienen una lista (vector en C++) para rastrear los nodos visitados en orden de anchura y evitan ciclos visitando un nodo solo una vez.
4. Imprimen o almacenan los nodos visitados en el orden en que se visitan, lo que representa el árbol en anchura resultante.

En resumen, en cuanto al algoritmo de conversión de grafo a árbol en anchura, ambos programas siguen el mismo enfoque fundamental del algoritmo BFS. La principal diferencia radica en la implementación general del programa y en cómo se ingresa y visualiza la información del grafo.

Mi programa:

```
/*================================================================
================
  == RECORRIDO DE GRAFO EN ANCHURA
==============================================
  ----------------------------------------------------------------
---------------
            > Juan Francisco Gallo Ramírez I.C.I.
================================================================
==============*/
#include <iostream>
#include <vector>
#include <queue>
#include <windows.h>
#include <locale.h>

using namespace std;

const char C = 2, FF = 187;

const int NEG = 0, AZU = 1, VER = 2, CEL=3, ROJ = 4, MOR = 5, AMA =
6, GRI = 7;
const int NEGC = 8, AZUC = 9, VERC = 10, CELC = 11, ROJC = 12, MORC =
13, AMAC = 14, GRIC = 15;

//---------------------------- FUNCIÓN PARA MANIPULAR EL COLOR DE LOS
CARACTÉRES ---//
void color(int c_fondo,int c_texto)
{
            int c_pix;
            c_pix = c_fondo*(16) + c_texto;
            HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
            SetConsoleTextAttribute(hConsole, c_pix);
}

//------------------------------------- FUNCIÓN PARA AGREGAR
ELEMENTOS AL VECTOR ---//
void agregar(vector<vector<string> >&vec, int x)
{
            string nodo;

            color(GRIC, AZU); cout<<"\t------------------------------
-----------------------" << endl;
            color(GRIC, AZU); cout<<"\t   "<<FF<<" Ingresa el nodo "
<< x+1 << " (nombre): ";
            cin >> nodo;
            vec[x].push_back(nodo);
            do
            {
                        color(GRIC, NEG); cout<<"\t     "<<FF<<" El
nodo " << vec[x][0] << " conecta con ('-' para terminar): ";
                        color(GRIC, AZU); cin >> nodo;
                        if(nodo != "-")
                        {
                                    vec[x].push_back(nodo);
                        }
            }while(nodo != "-");

}

//------------------------------------- FUNCIÓN IMPRIMIR LA LISTA DE
ADYACENCIA ---//
void imprimirLista(vector<vector<string> >& vec)
{

    for (int i = 0; i < vec.size(); i++)
            {
        color(GRIC, NEG); cout << "\t   Nodo " << vec[i][0] << ": ";
        for (int j = 1; j < vec[i].size(); j++)
                        {
            color(GRIC, AZU); cout << vec[i][j] << " ";
        }
        cout << endl;
    }
}

//----------------------- FUNCIÓN PARA DEVOLVER EL INDICE DE UN
ELEMENTO BUSCADO ---//
int buscarNodo2(vector<vector<string> > vector, string nodo)
{
            for (int i = 0; i < vector.size(); i++)
            {
        if(vector[i][0] == nodo)
        {
            return i;
                        }
    }
    return -1;
}

//----------------------- FUNCIÓN PARA DEVOLVER EL INDICE DE UN
ELEMENTO BUSCADO ---//
int buscarNodo1(vector<string> vector, string nodo)
{
            for (int i = 0; i < vector.size(); i++)
            {
        if(vector[i] == nodo)
        {
            return i;
                        }
    }
    return -1;
}

//---------------------------- FUNCIÓN PARA DOBTENER LA ANCHURA DE UN
GRAFO DADO ---//
void obtenerAnchura(vector<vector<string> >& vec, queue<string>
&cola, vector<string> &anchura)
{
            string nodo;
            int i_nodo;

            color(GRIC, AZU);cout<< endl << "\t ----------------------
----------------------" << endl << endl;
            color(GRIC, AZU); cout << "\t   " << FF; color(GRIC,
NEG); cout << " Ingresa el nodo de partida (nombre): ";
            color(GRIC, AZU); cin >> nodo;
            i_nodo = buscarNodo2(vec, nodo);

            anchura.push_back(vec[i_nodo][0]);
            cola.push(vec[i_nodo][0]);

            while(!cola.empty())
            {
                        cola.pop();
                        i_nodo = buscarNodo2(vec, nodo);
                        for(int i=1; i<vec[i_nodo].size(); i++)
                        {
                                    if(buscarNodo1(anchura,
vec[i_nodo][i]) == -1)
                                    {

            cola.push(vec[i_nodo][i]);

            anchura.push_back(vec[i_nodo][i]);
                                    }
                        }
                        if(!cola.empty())
                                    nodo = cola.front();
            }

            color(GRIC, AZU);cout << endl <<"\t === RECORRIDO EN
ANCHURA ================" << endl << endl;
            for (int i = 0; i < anchura.size(); i++)
            {
        color(GRIC, NEG); cout << "\t\t" << anchura[i] << endl;
    }
}

//------------------------------------------------------------
FUNCIÓN PRINCIPAL ---//
int main()
{
            int cant_nod;
            float prom;
            queue<string> cola;
            vector<string> anchura;

            setlocale(LC_ALL, "");system("color F0");
            color(AZU, GRIC); cout<<"
RECORRIDO DE GRAFO EN ANCHURA                          " << endl <<
endl;
            color(GRIC, AZU); cout << "\t" << FF; color(GRIC, NEG);
cout << " Ingresa la cantidad de nodos del grafo: ";
            color(GRIC, AZU); cin >> cant_nod; cout << endl;

            //7 A - B H - C R - D B C - H A D T - R H - T -
            vector <vector<string> > vec(cant_nod);

            for(int i=0; i<cant_nod;i++)
            {
                        agregar(vec, i);
                        cout<<endl;
            }


            system("cls");
            color(AZU, GRIC); cout<<"
RECORRIDO DE GRAFO EN ANCHURA                          " << endl <<
endl;
            color(GRIC, AZU); cout<<"\t === LISTA DE ADYACENCIA
==================" << endl << endl;
            imprimirLista(vec);
            obtenerAnchura(vec, cola, anchura);

            return 0;
}
```

Programa de Chat GPT:

```
#include <iostream>
#include <vector>
#include <queue>

using namespace std;

class Grafo {
public:
    Grafo(int vertices);
    void agregarArista(int v, int w);
    void convertirEnArbol(int nodoRaiz);

private:
    int vertices;
    vector<vector<int> > listaAdyacencia;
};

Grafo::Grafo(int vertices) {
    this->vertices = vertices;
    listaAdyacencia.resize(vertices);
}

void Grafo::agregarArista(int v, int w) {
    listaAdyacencia[v].push_back(w);
    listaAdyacencia[w].push_back(v); // Si el grafo no es dirigido, agrega esta línea.
}

void Grafo::convertirEnArbol(int nodoRaiz) {
    vector<bool> visitado(vertices, false);
    queue<int> cola;

    visitado[nodoRaiz] = true;
    cola.push(nodoRaiz);

    while (!cola.empty()) {
        int actual = cola.front();
        cout << actual << " ";
        cola.pop();

        for (int i = 0; i < listaAdyacencia[actual].size(); ++i) {
            int vecino = listaAdyacencia[actual][i];
            if (!visitado[vecino]) {
                visitado[vecino] = true;
                cola.push(vecino);
            }
        }
    }
}

int main() {
    Grafo g(7); // Cambia el número de vértices según tu grafo

    g.agregarArista(0, 1);
    g.agregarArista(0, 2);
    g.agregarArista(1, 3);
    g.agregarArista(1, 4);
    g.agregarArista(2, 5);
    g.agregarArista(2, 6);

    cout << "Árbol en anchura a partir del nodo 0:" << endl;
    g.convertirEnArbol(0);

    return 0;
}
```