



Proyecto Final Algorítmica


Integrantes: Gabriel Neme

Pablo Badani

Richard Rojas

Miguel Molina

Repositorio GitHub

 GaboRex / BFS-JAMBAO

Unwatch 1

Unstar 1

Fork 0

<> Code

Issues

Pull requests

Actions

Projects

Wiki

Security

Insights


Settings

main 2 branches 0 tags

Go to file

Add file


Code

 GaboRex actualizando...

1 2889323 yesterday 41 commits

.vscode	Merge branch 'main' of https://github.com/GaboRex/BFS-JAMBAO	2 days ago
pruebas	Carpeta de pruebas	yesterday
README.md	actualizando...	yesterday
bfs.c++	hola	yesterday
bfs.exe	hola	yesterday

☰ README.md



```
printf("Mucha Cumbia");
```

About

Hola mundos por doquier y mucha cumbia tambien.

Readme

Releases

No releases published
[Create a new release](#)

Packages

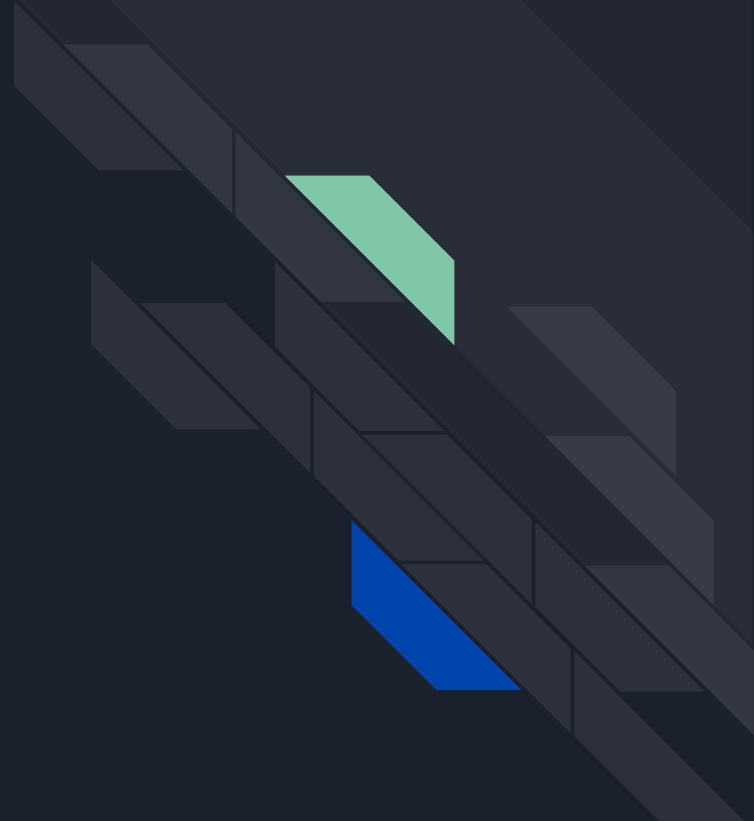
No packages published
[Publish your first package](#)

Environments 1

[aithub-pages](#) Active

Configuración

- Algoritmo BFS.
- Requisitos de Software.
- Cómo ejecutar el programa.





Proceso de Instalación

- Instalar un editor de texto con compilador o en el mejor de los casos un IDE con el que debemos trabajar el programa
- Si no se cuenta con los requisitos para ejecutar el programa, se puede utilizar un compilador online.
- Crear una carpeta en donde guardaremos el código trabajado
- Crear un repositorio en la página GitHub.
- Public (Pública) predeterminada, cualquiera podrá ver este repositorio o Private (Privado) para limitar quién puede ver el repositorio y confirmarlo(esto no es gratis).
- Desactive la casilla de verificación Initialize this repository with a README (Inicializar este repositorio con un archivo README). En su lugar creará un archivo README.md manualmente en el siguiente paso. Elija "Create repository" crear repositorio.

Algunos comandos importantes:

COMANDOS BÁSICOS DE git



GIT es un sistema de control de versiones que nos ayuda a llevar el historial completo de modificaciones de un proyecto.

Todo desarrollador sin importar el lenguaje **debe dominar Git**.
Prof. Betó Quiroga

GIT INIT



Inicia un nuevo repositorio.

GIT ADD



Añade un archivo a la zona de montaje. **git add *** añade uno o más archivos a la zona de montaje.

GIT LOG



Se utiliza para listar el historial de versiones de la rama actual.

GIT RESET



Descompone el archivo, pero conserva el contenido del mismo.



GIT CLONE

Clona un repositorio existente.

GIT CONFIG



Establece el nombre del autor, el correo y demás **parámetros** que Git utiliza por defecto.

GIT STATUS



Enumera todos los archivos que deben ser confirmados.

GIT DIFF



Muestra las diferencias de archivo que aún no se ponen en escena.

COMANDOS BÁSICOS DE git (PARTE 2)

Si quieres trabajar en el mundo del desarrollo de software, es obligatorio que sepas GIT.

GIT HELP



Muestra información para saber el uso de cada comando de git.

GIT CLEAN



Elimina archivos no deseados de un repositorio.

GIT BRANCH



Muestra la lista de ramas que existen en un repositorio.

GIT MERGE



Fusiona dos o más ramas.

GIT PULL



Descarga y actualiza los cambios realizados desde un repositorio remoto a tu repositorio local.

GIT PUSH



Sube los archivos a un repositorio remoto.

GIT CHECKOUT



Sirve para moverse de una rama a otra y regresar en el tiempo.

GIT REMOTE



Crea, visualiza y elimina conexiones a otros repositorios.



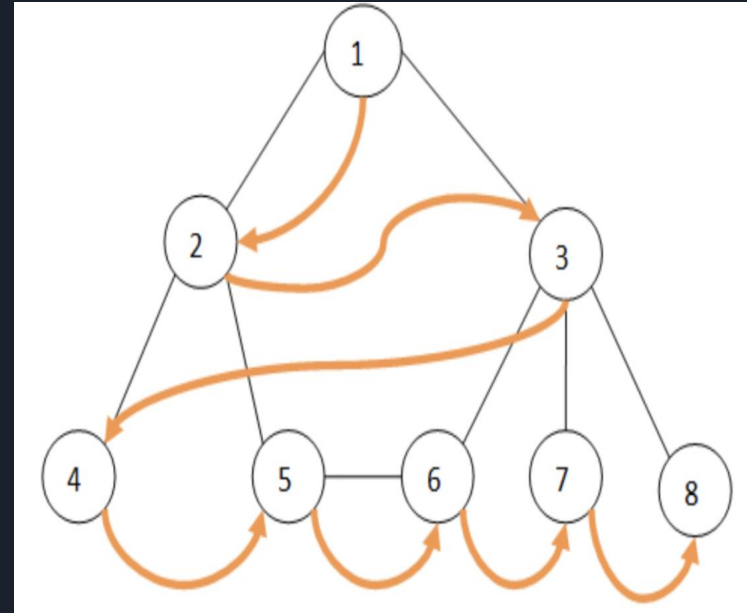
Definición del problema

Nuestro programa construye una previsualización del grafo a partir de sus entradas, ayudando a que sea más sencillo visualizar las conexiones entre servidores.

El algoritmo implementado va ayudar a que las personas conozcan un tiempo estimado de llegada de un servidor a otro, verificando si hay una conexión entre el punto inicial y el punto final.

Explicación del Algoritmo - BFS

Búsqueda en Anchura (BFS) es un algoritmo para recorrer o buscar estructuras de datos de árbol o gráfico . Comienza en la raíz del árbol y explora todos los nodos vecinos en la profundidad actual antes de pasar a los nodos del siguiente nivel de profundidad.





Explicación del Algoritmo

```
//FUNCION PARA MOSTRAR EL GRAFO AL INGRESAR LOS SERVIDORES Y CONEXIONES
void mostrarElGrafo(int servidores){
    cout<<"El grafo es: "<<endl;
    for(int i=1;i<=servidores;i++) {
        cout<<"{"<<(i)<<"}=>";
        for(int j=0;j<grafo[i].size();j++) {
            cout<<"["<<grafo[i][j]<<"]";
        }
        cout<<endl;
    }
}
```


Explicación del Algoritmo

```
//MUESTRA EL GRAFO EN UNA MATRIZ DE ADYACENCIA
void mostrarLaMatriz(int servidores, int conexiones){
    int matrizAdyacencia[servidores][servidores];
    for(int l = 0; l<servidores; l++){
        for(int h = 0; h<servidores;h++){
            matrizAdyacencia[l][h] = 0;
        }
    }
    for(int i=0;i<conexiones;i++) {
        cout<<"Ingrese la conexion entre dos servidores, esta conexion sera mutuo"<<endl;
        int a,b;
        cin>>a>>b;
        matrizAdyacencia[a-1][b-1] = 1;
        matrizAdyacencia[b-1][a-1] = 1;
    }
    cout<<"La matriz es: "<<endl;
    cout<<" 1 ";
    for(int i=2;i<=servidores;i++) {
        cout<<i<<" ";
    }
    for(int j=1;j<=servidores;j++) {
        cout<<endl<<j<<" ";
        for(int n=0;n<servidores;n++) {
            cout<<" ["<<matrizAdyacencia[j-1][n]<<"] ";
        }
        cout<<endl;
    }

    cout<<endl;
}
```

Explicación del Algoritmo

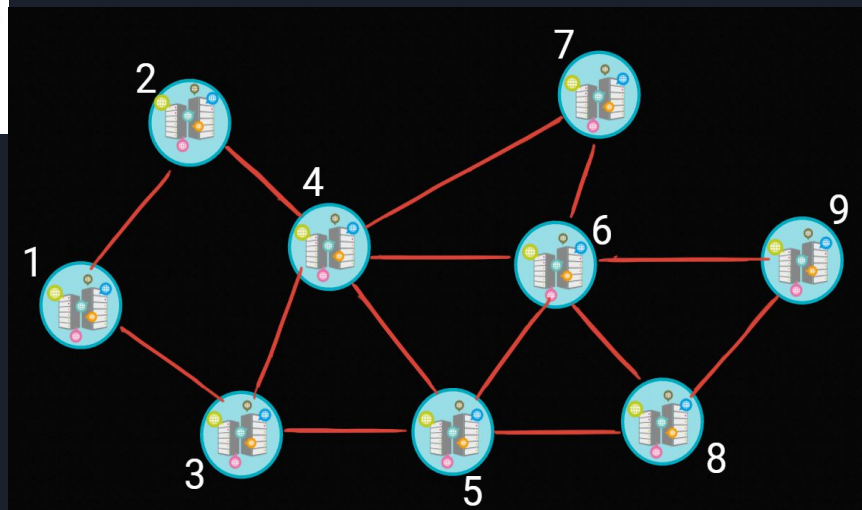
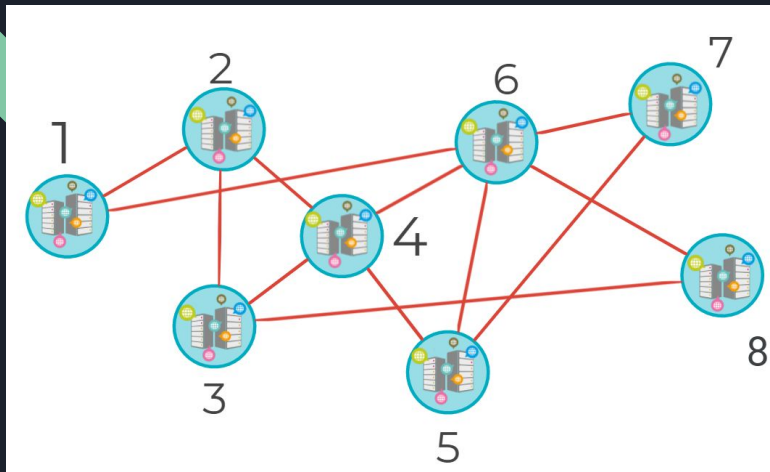
//MUESTRA EL RECORRIDO HECHO POR LOS SERVIDORES PARA TRANSPORTAR EL PAQUETE DE DATOS

```
void mostrarRecorridoBFS(int nodoActual, int servidor, int nodoFinal){
    int nodoInicial = nodoActual;
    vector<pair<int, int>> padres;
    vector<bool> visitados(servidor+1);
    queue<int> colita;
    visitados[nodoActual] = true;
    colita.push(nodoActual);
    vector<int> level(servidor+1);
    level[nodoActual] = 0;
    while(!colita.empty() && !visitados[nodoFinal]){
        int nodoActual = colita.front();
        colita.pop();
        for(int i=0; i<grafo[nodoActual].size(); i++) {
            int nodoAVisitar = grafo[nodoActual][i];
            if(!visitados[nodoAVisitar]) {
                padres.push_back(make_pair(nodoActual, nodoAVisitar));
                colita.push(nodoAVisitar);
                visitados[nodoAVisitar] = true;
                level[nodoAVisitar] = level[nodoActual] + 1;
            }
        }
    }
}
```

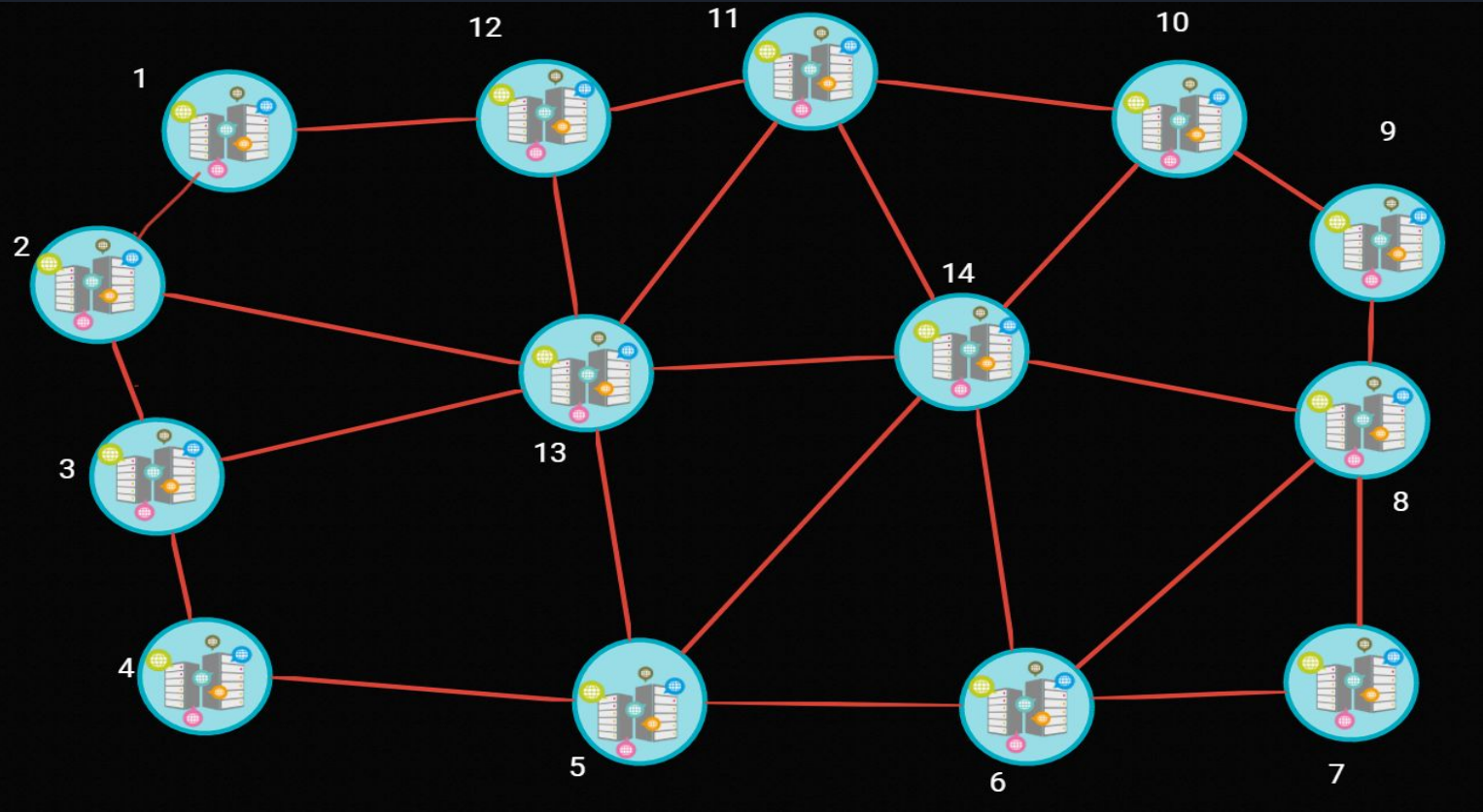
```
if(visitados[nodoFinal-1]){
    cout<<"el paquete de datos si llego"<<endl;
}
else{
    cout<<"el paquete de datos no llego :( " <<endl;
}
vector<pair<int, int>> trayectoBFS;
trayectoBFS.push_back(padres[padres.size()-1]);
pair<int,int> actuales = padres[padres.size()-1];
while(trayectoBFS[trayectoBFS.size()-1].first != nodoInicial){
    for (int i = 0; i < padres.size(); i++) {
        if(actuales.first == padres[i].second) {
            trayectoBFS.push_back(padres[i]);
        }
    }
    actuales = trayectoBFS[trayectoBFS.size()-1];
}
for(int f = trayectoBFS.size()-1; f > 0; f--){
    cout<<trayectoBFS[f].first<<"--"<<trayectoBFS[f].second<<endl;
}
cout<<trayectoBFS[0].first<<"--"<<trayectoBFS[0].second<<endl;

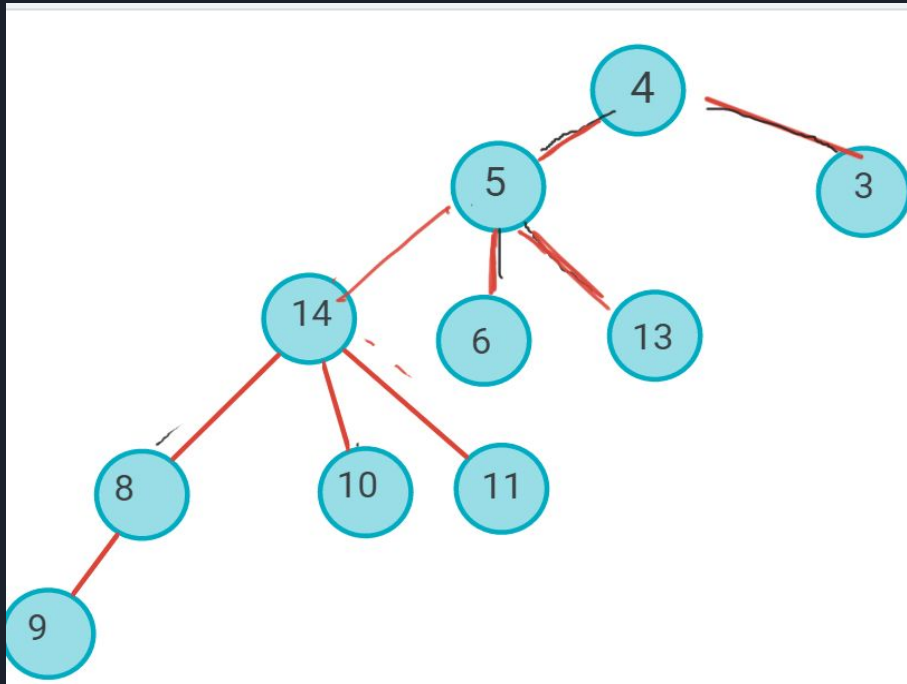
cout<<trayectoBFS.size()<<endl;
```

Ejemplos

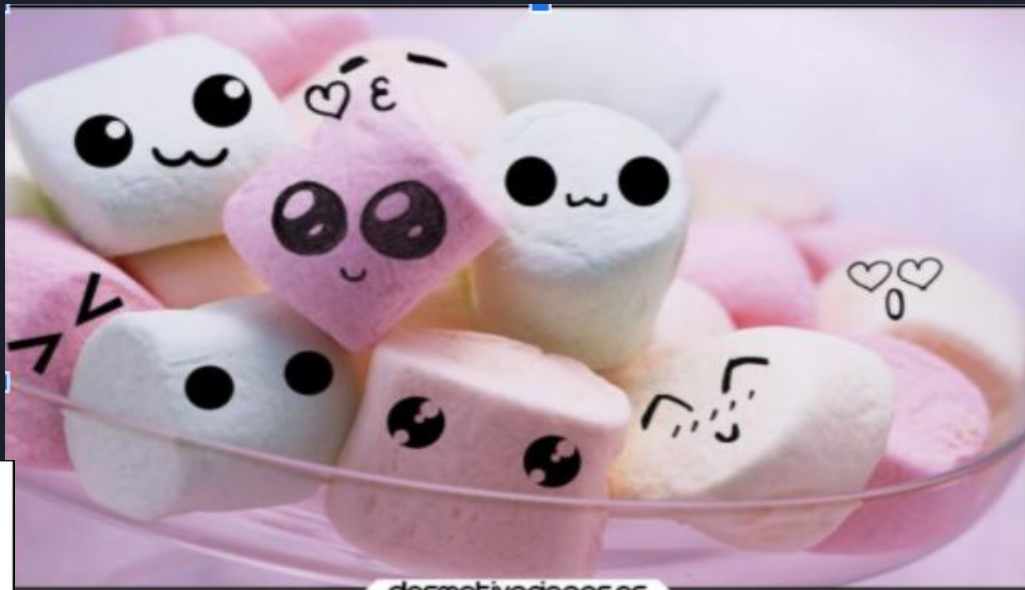


Ejemplos.





Arboles de Expansion



Gracias por su atención!!

Si es que la prestarón

