بسم الله الرحمن الرحیم

نام : عماد

نام خانوادگی : آقاجانی

شماره دانشجویی : 88521344

تکلیف : پیاده سازی پیمایش گراف

درس : ساختمان گسسته

IDE : Microsoft Visual Studio 2008

Farsi Font : BNazanin

فهرست مندرجات :

1. توضیح‌ فایل‌های ارسال شده
2. توضیح الگوریتم های BFS و DFS .
3. فایل های ارسال شده :
   * 1. BFS\_DFS\_OOP
     2. BFS\_DFS

مورد i : بصورت Object Oriented نوشته شده است. کلاس اصلی شامل یک vector<vector<bool>> data; است که ماتریس مجاورت گراف را در خود نگه می‌دارد و شامل توابع عضو زیر است :

private:

vector<bool>& operator[](int);

int GSIZE; //store Graph size

public:

Graph();

void showMatrix();//Print Matrix

void load(char\*); //load from file

void load(void); //load manulally

void BFS();

void DFS();

مورد ii : این کد ساده شده‌ی مورد i بوده و بصورت ماژولار نوشته شده است .

1. توضیح الگوریتم BFS و DFS ( توضیحات از روی فایل BFS\_DFS.cpp که بصورت OOP نیست، داده میشود )

در مرحله اول یک ماتریس مجاورت از گراف که در اینجا بصورت دینامیک پیاده‌سازی شده ، ایجاد می کنیم .

vector<vector<bool>> graph;

makeGraphDynam(graph); //\*\*

const int GRAPHSIZE=graph.size();

bool \*mark=new bool[GRAPHSIZE];

for(int i=0;i<GRAPHSIZE ;i++)

mark[i]=false;

که تابع makeGraphDynam بصورت زیر پیاده‌سازی شده ..

void makeGraphDynam(vector<vector<bool>> &graph) //\*\*

{

graph.clear();

int a=0,b=0;

ifstream input("input.txt");

if(input)

while(input>>a>>b)

{

int temp = a>b?a:b;

if(temp > graph.size())

{

graph.resize(temp);

for(int i=0;i<temp;i++)

graph[i].resize(temp);

}

(graph)[a-1][b-1]=(graph)[b-1][a-1]=true;

}

input.close();

}

در صورتی که بخواهیم دستی یال ها را مشخص کنیم ، کافیست ..

cout<<"Enter your pair vertex and end it with 0 0 pair \n";

int a=0,b=0;

while(cin>>a>>b && ( a!=0 || b!=0) )

{

…

}

**BFS**:

queue<int> myQu;

myQu.push(1);

mark[0]=true;

while(!(myQu.empty()))

{

for(int j=1;j<=GRAPHSIZE;j++)

{

if( graph[myQu.front()-1][j-1] && !mark[j-1] )

{

myQu.push(j);

mark[j-1]=true;

cout<<"Connect :: Parent: "<<myQu.front()<<" Child: "<<j<<endl;

}

}

myQu.pop();

}

for(int i=0;i<GRAPHSIZE ;i++)

if(!mark[i])

{

cout<<"\nYour Graph is not Connective ! \n";

break;

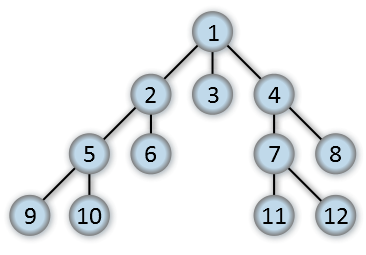
}

در بخش اول یک صف که اساس کار الگوریتم BFS است ، ساخته شده و راس شماره 1 را بعنوان نقطه شروع پیمایش بر روی گراف در نظرگرفته ایم و مقدار mark[1-1] را نیز True قرار می‌دهیم . آرایه <bool> mark برای علامتگذاری راس های پیمایش شده استفاده می‌شود.

روند کلی این الگوریتم به این صورت است که، راس هایی که به صف وارد می‌شوند بعنوان راس‌های پیمایش شده در نظر گرفته می‌شود .

سپس یک راس پیمایش شده از سر صف خارج و یکسری راس پیمایش نشده بصورتی که توضیح داده خواهد شد، وارد می‌شود .

سطی یا عمقی بودن اینگونه پیمایش ها بواسطه نوع Data Structure ی که در پیاده سازی استفاده شده ، صورت می‌گیرد .



توجه شود که شماره اندیس ماتریس مجاورت و ارایه mark هردو یک واحد از شماره راس‌ها کمتر است ولی مقدار هایی که در صف و پشته قرار می‌دهیم برابر خود مقدار عددی شماره راس های گراف است !

در بخش دوم الگوریتم مراحل زیر، تا پایان پیمایش، تکرار می‌شود :

1. از سر صف یک راس ( که بدیهتا ، یک راس پیمایش شده است ) برداشته می‌شود.
2. تمام راس های متصل به این راس که قبلا پیمایش نشده ، به انتهای صف اضافه می‌شود و با mark[j-1]=true بعنوان یک راس پیمایش شده در نظر گرفته می‌شود .

اگر در مرحله ای صف خالی شود ، بدین معناست که هیچ راسی باقی نمانده که به یکی از راس های پیمایش شده متصل باشد ولی پیمایش نشده باشد . در این مرحله پیمایش تمام شده و اگر گراف یکپارچه و همبند باشد ، گراف کامل پیمایش شده است .

در مرحله آخر چک می‌شود که راس پیمایش نشده ای باقی مانده یا نه، که در صورت وجود بمعنی ناهمبند بودن گراف است .

**DFS**:

stack<int> myStack;

for(int i=0;i<GRAPHSIZE ;i++)

mark[i]=false;

myStack.push(1);

mark[0]=true;

while(!(myStack.empty()))

{

int myTop=myStack.top();

myStack.pop();

for(int j=GRAPHSIZE;j>0;j--)

{

if( graph[myTop-1][j-1] && !mark[j-1] )

{

myStack.push(j);

mark[j-1]=true;

cout<<"Connect:: Parent: "<<myTop<<" Child: "<<j<<endl;

}

}

}

در بخش اول یک پشته ( که در عمل یک ارایه دینامیک است که تغییرات محتوایی تنها از یک طرف صورت میگیرد ) که اساس کار الگوریتم DFS است ، ساخته شده و راس شماره 1 را بعنوان نقطه شروع پیمایش بر روی گراف در نظرگرفته ایم و مقدار mark[1-1] را نیز True قرار می‌دهیم .

روند کلی این الگوریتم مانند BFS است و راس هایی که به پشته وارد شود ، راس های پیمایش شده در نظر گرفته می‌شود . Data Structure پشته باعث می‌شود که نوع پیمایش عمقی باشد .

در بخش دوم مراحلی مشابه BFS را داریم ..

1. از بالای پشته یک راس ( که بدیهتا ، یک راس پیمایش شده است ) برداشته می‌شود.
2. تمام راس های متصل به این راس که قبلا پیمایش نشده ، به بالای پشته اضافه می‌شود و با mark[j-1]=true بعنوان یک راس پیمایش شده در نظر گرفته می‌شود .

در DFS بعلت اینکه ورود و خروج از یک طرف است و همچنین ما تنها به بالاترین عنصر پشته دسترسی داریم ، ترتیب مراحل بالا خیلی مهم است. ( در حالی که در صف (و درنتیجه BFS ) اینچنین نبود )

در پایان، اگر در مرحله ای پشته خالی شود ، بدین معناست که هیچ راسی باقی نمانده که به یکی از راس های پیمایش شده متصل باشد ولی پیمایش نشده باشد . در این مرحله پیمایش تمام شده و اگر گراف یکپارچه و همبند باشد ، گراف کامل پیمایش شده است .