**پروژه ی دوم درس ساختمان داده ها**

# طراحی خودکاره های حالت متناهی

هدف اصلی ما در این پروژه ، بیان پذیرش یا عدم پذیرش رشته هاست .

برای راحتی کار از همان تعریف خود پروژه استفاده شده است .( جفت "آ" و تعداد دلخواه "ب")

حالتهای مجاز حالتهایی هستند که به 1 ختم شوند.



کلاس ها :

1. Node:

هر گره لازم دارد بداند که با چه چیزی به کجا برود . داخل این کلاس لیستی از یال ها تعریف میکنیم که برای دی اف اس زدن راحت تر باشیم! اینگونه در نهایت برای یافتن همسایگی ها راحت تر خواهیم بود .

Array List<Edge> edge list =new Array List<>(); //listi az yaal haa, baraye dfs zadan rahat tare k bdoone ki baa ki hamsaye hast

Boolean visit; //hamoon flagi k baraye dfs lazem darim

Boolean isok;  
int value;

1. Graph:

چون ما نهایتا میخواهیم خروجی مان به صورت یک گراف دربیاید ، احتیاج به یک کلاس گراف داریم .

هر گراف مجموعه ای از نود ها و یال هاست . برای اینکه اطلاعات نود ها و یال ها را نگهداری کنیم ، برای هر یک کلاسی جداگانه مینویسیم.

تعداد نود ها را نیز در گراف ذخیره میکنیم.

Array List<Node> node list = new Array List<>() ; //listi az node haa

1. Control:

قسمت اصلی کارمان این است که مشخص کنیم چه کارهایی میخواهیم انجام بدهیم .

خواندن از ورودی و تبدیل کردن اطلاعات به گراف یکی از کارهاست گه کلاس گراف باید آن را انجام دهد .

کار دوم آن نمایش گراف است که به صورت گرافیکی خواهد بود .(به کمک graph viz)

کار بعدی آن رودی گرفتن به صورت رشته و چک کردن آن هاست .

Boolean cycle=false; //baraye tashkhise vojoode dor dar geraph  
Edge cycle\_edge; //az type yaal   
Node cycle\_node; //az type node

Create graph

Print Graph

Is Valid

Is Cyclic

DFS

Delete Cycles

Is Valid

برای حالت هایی که پیش بینی نکرده بودیم ، یک متغیر بولین را ابتدا با مقدار فالس قرار میدهیم ، که برایمان چک کند که اصلا پیدا کرد حالت مورد نظر را ، یا نه! اگر پیدا نکرد دیگر کلا بررسی نخواهد شد !

در داخل حلقه هر بار این متغیر را دوباره فالس میکنیم و در آخر بیرون حلقه هم یک شرط میگذاریم که اگر مجاز بود که کاری به آن نداریم و به سر حلقه برمیگردیم در غیر این صورت مقدار فالس را برمیگرداند .

ما یک پرچم دیگر هم میخواهیم ، برای مواقعی که مثلا یک نود را تا عمق کامل چک کرده ایم و هیچ دوری نداشته ، دیگر کاری به آن نود نداریم و میتوانیم بررسی اش نکنیم (اگر اینکار را نکنیم ممکن است داخل یک لوپ بینهایت بیفتیم و نودهای تکراری پیدا کنیم ) ,.)

boolean valid= false;

valid=false;

1. Edge
2. int nextnode; //node i ke behesh vasle(node nahaii)
3. char value; //meghdare yaal

5) Main

# ویژگی ها

***:تعریف خودکاره ی حالت متناهی***

به صورت تعریف کردن ویژگی ها ی مورد نظر هر گراف(یال ها و نودها) انجام میگیرد

Edge

Node

Graph

***.***

***گرفتن لیست خودکاره ها:***

Create graph

***نمایش گرافی***

Print Graph

***تشخیص عضویت رشته ی وارد شده:***

Is Valid

***تشخیص وجود دور***

Is Cyclic 🡺 از دی اف اس استفاده میکند !

DFS

وقتی هنگام چک کردن، دوباره به نودی برسید که قبلا آن را چک کرده بودید دور ایجاد شده است .

پس نیاز به یک پرچم نشانه داریم که به ما اعلام کند این نود قبلا ویزیت شده است یا خیر ! (برای راحتی کار دو پرچم در نظر می

گیریم، یکی برای آن که بدانیم ویزیت شده یا نه و دیگری برای آن که بدانیم از کدام نود به کدام رفته ایم و برای حذف دور ها کارمان راحت شود .)

***دی اف اس را به طور بازگشتی روی آن صدا میکنیم .***

برای قسمت بازگشتی در دی اف اس زدن ، من هنگام مواجهه با گراف طوقه دار به مشکل خوردم . مشکل آنجا بود که هنوز دی اف اس تمام نشده بود در حالی که به خود نود میرسیدم و باعث میشد یک یال اضافه حذف شود . (یک حالت کاملا استثنایی).

به همین جهت این حالت خاص را جدا بررسی کردم .

If (node.value == next.value)   
{  
 cycle\_edge=node.edgelist.get(i);  
 cycle\_node = node;  
 cycle=true;  
 return;  
}

***حذف دور :***

Delete Cycles

از تکنیک تشخیص دور که قبلا نوشته بودیم استفاده میکنیم و آنرا به عنوان شرط برای این تابع در نظر میگیریم .

**پس از حذف دورهای موجود در خودکاره، یک خودکاه ی جدید خواهیم داشت که از لحاظ قدرت پردازش و محاسبه و ویژگی های دیگر با خودکاره ی اولیه متفاوت است .**