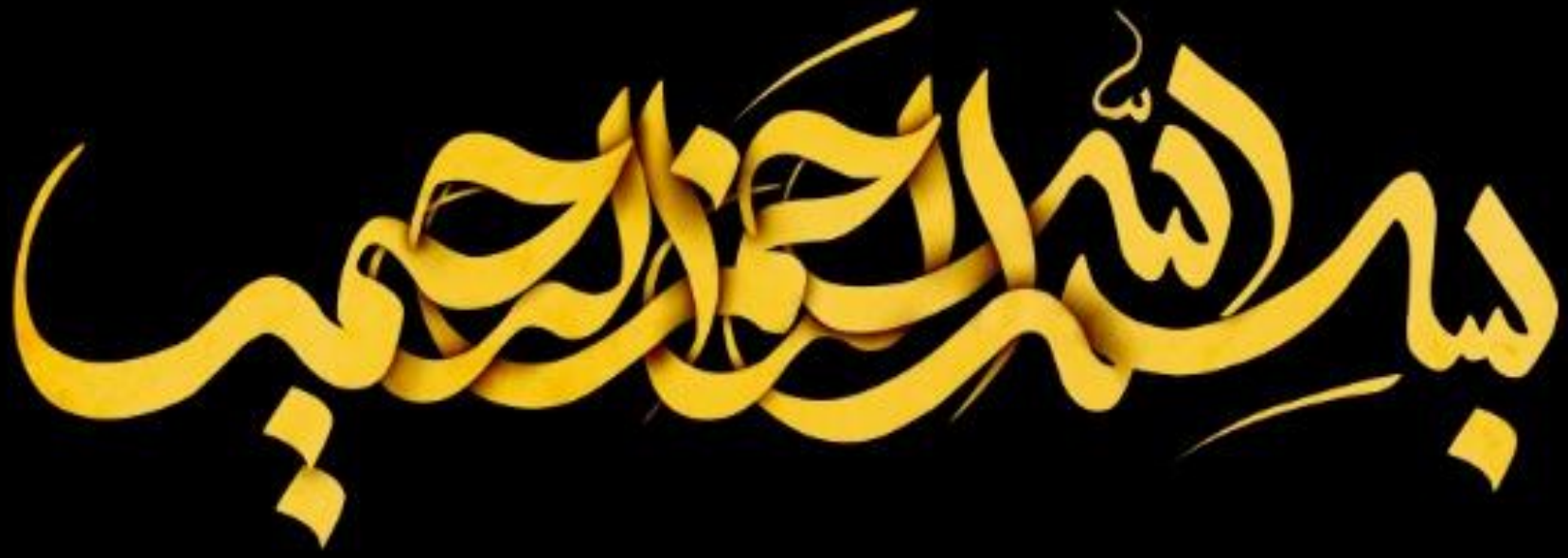
بسم الله الرحمن الرحیم

گراف در Sage

الهام ایزدی زاده 9303784

رسم گراف

```
d={0:[1,4,3],1:[4,2],2:[0],5:[ ]}
G=Graph(d)
G.plot()
```

حال اگر موقعیت راس‌ها مهم باشد از دستور زیر استفاده می‌کنیم.

```
Pos={0:[0,0],1:[1,2],2:[-1,1],3:[1,-3],4:[2,5],5:[-1,-1]}
G.graphplot(pos=pos).plot()
```

با اضافه کردن راس و یا یال جدید موقعیت راس ها تغییر می‌کند. برای ثابت ماندن راسها، این کار را میکنیم

```
L=G.graphplot(pos=pos)
```

حال تغییرات را روی L انجام میدهیم.

برای اینکه موقعیت راس‌ها را ببینیم از دستور زیر استفاده می‌کنیم

```
G.get_pos()
```

اما برای گراف هایی که خودمان طراحی کردیم این دستور موقعیت راسها را نمیدهد باید اول نمیدهد باید اول موقعیت راس ها را ثابت کنیم بعد این دستور را بنویسیم.

```
m=Graph({0:[1,4],1:[2],2:[3],3:[4]})
m.plot(save_pos=True)
r=m.get_pos()
Show(r)
```

وقتی موقعیت راس ها را خودمان بدهیم این دستور برای موقعیت راس‌ها None چاپ میکند.

اضافه کردن راس

```
G.add_vertices([10,11,12])
```

اضافه کردن یال

```
G.add_edges([(1,10),(30,8)])
```

اگر بخواهیم یک یال چند گانه برای گراف چند گانه اضافه کنیم برای اینکه یک یال را چند بار تکرار نکنیم از دستور زیر استفاده می‌کنیم.

```
G.add_edges([(0,1)]*3)
```

نشان دادن راس‌های یک گراف

G.vertices()

نشان دادن یال‌های یک گراف

G.edges()

تعداد یال‌های یک گراف(اندازه گراف)

G.size()

تعداد راس‌های یک گراف

G.order()

نشان دادن همسایه‌های یک راس(بدون جهت)

G.neighbors()

(با جهت)

G.neighbors_in() & G.neibors_out()

نشان دادن برچسب یال

```
G=Graph({0:{1:'x',2:'z',3:'a'},2:{5:'out'}})

G.edge_ labels()
```

که خروجی مورد نظر {x, z, a, out} است.

برای نشان دادن برچسب یک یال از دستور G.edge_label(0,1) استفاده می‌کنیم.

اضافه کردن حلقه به یک گراف

```
K=graphs.PetersenGraph()
K.allow_loops(True)
K.add_edge(0,0)
K.show()
```

تغییر پیش فرض اندازه گراف رسم شده

Sage.graphs.graph_plot.DEFAULT_SHOW_OPTIONS['figsize]= [3,3]

حال اگر تغییر اندازه یک گراف مد نظر باشد نه تغییر پیش فرض از این شیوه استفاده می‌شود

Graphs.PetersenGraph().show(figsize=[3,3])

تغییرویژگی‌های مورد نظربرای رسم گراف

تغییر سایز راس گراف	vertex_size=200
اجازه برچسب دادن به راس یک گراف	vertex_labels=True
رنگ یال یک گراف	edge_color='blue'
طرح یال یک گراف	edge_style='solid'(dotted,dashed,dashdot)
اجازه برچسب دادن به یال یک گراف	edge_labels=False
جهت ریشه یک درخت	tree_orientation='down'(up,left,right)
کادر کشیدن دور گراف	graph_border=false
نشان دادن راس‌ها به صورت سفید و بزرگ	talk=False

فاصله بین چند یال dist=0.075

بیشترین فاصله بین چند یال max_dist=1.5

اندازه شعاع حلقه loop_size=0.75

بعد dim=2 &3

رنگ یال‌های با برچسب‌های مساوی مثل هم باشد color_by_label

ویژگی‌های مورد نظر را داخل پرانتز به صورت زیر یادداشت کنید

```
G=graphs.PetersenGraph()
    G.plot(vertex_size=200, ...)
```

برچسب دادن به گراف و کشیدن یال چند گانه

```
B=Graph({},loops=True,multiedges=True)

B.add_edges([(0,0,'a'),(0,0,'b'),(0,1,'d'),(0,1,'b'),(2,1,'g')])

GP=B.graphplot(edge_labels=True,color_by_label=True)

GP.plot()
```

تغییر در یال‌ها

```
GP.set_edges(edge_style='dotted')
```

تغییر در راس‌ها

```
GP.set_vertices(vertex_size=100 )
```

تغییر رنگ دریال‌های مورد نظر

```
W=graphs.DodecahedralGraph()
W.show(edge_colors={‘red’:[(6,5),(2,3)],(0,1,1):[(0,1)]})
```

همان‌طور که دیدید می‌توان به جای اسم رنگ از ترکیب رنگی آن یعنی (0,1,1) استفاده کرد.

گراف جهت دار

```
R=DiGraph({0:[0,1],1:[2],2:[3],3:[2]},loops=True)
R.show()
```

گراف جهت دار چند گانه

```
Q=DiGraph({},loops=True,multiedges=True)
Q.add_edges([(0,0,'a'),(0,1,'t'),(0,1,'y'),(1,2,'z')])
Q.plot(edge_labels=true)
```

درخت

```
E={1:[2,3],2:[4,5],3:[6,7]}

P=Graph(E)

P.plot(layout='tree',tree_orientation='up', tree_root=2)
```

درخت پوشا = یک دخت پوشا T از گراف همبند و بدون جهت G درختی است، که شامل تمام رئوس و حداقل برخی یال‌ها می‌باشد.

درخت پوشا کمینه = درختی است که در بین درخت‌های پوشای آن گراف ،مجموع وزن یال‌های آن کمترین مقدار ممکن باشد.

```
H=graphs.HoffmanSingletonGraph()
T=Graph()
T.add_edges(H.min_spanning_tree(starting_vertex=0))
T.show(layout='tree',tree_root=0)
```

برای اینکه شکل گراف را بهتر ببینید از دستور figsize استفاده کنید

تعداد درختان پوشا

```
H.spanning_trees_count()
```

آیا گراف یال چند گانه دارد؟	q.has_multiple_edges()
آیا گراف میتواند یال چند گانه داشته باشه؟	q.allows_multiple_edges()
یال‌های چند گانه گراف.	q.multiple_edges()
آیا گراف حلقه دارد؟	q.has_loops()
آیا گراف می‌تواند حلقه داشته باشد؟	q.allows_loops()
حلقه‌های موجود در گراف.	q.loops()
ایا گراف یال مورد نظر را دارد؟	q.has_edge()

مسیر در گراف

```
F=graphs.PetersenGraph()
```

```
sorted(F.all_paths(1,4))
```
تمام مسیرها را به ما نشان می‌دهد

```
F.shortest_path(1,4)
```
کوتاه ترین مسیر بین دو تا راس

```
F.distance_matrix()
```
ماتریس فاصله بین رئوس

برای گراف جهت دار هم از همین دستور استفاده می‌شود.

ادغام کردن دو راس

```
G.merge_vertices((1,4))
```

تقسیم کردن یک یال به چند قسمت

بین یال (0و1) سه راس اضافه میکند

```
G.subdivide_edge(0,1,3)
```

اضافه کردن یک دور به گراف با دادن راس‌های آن

```
G.add_cycle([3,1,4])
```

برگرداندن طرح ساده یک گراف :یعنی اگر گراف دارای یال‌های چند گانه یا حلقه یا جهت دار باشد آنها را حذف میکند و طرح ساده گراف را بر می‌گرداند.

G.to_simple()

ماتریس مجاورت

```
U=graphs.CubeGraph(4)
U.adjacency_matrix()
```

چند جمله‌ای مشخصه گراف

```
U.characteristic_polynomial()
```

یا از دستور U.charpoly() استفاده میکنیم.

برای ماتریس مجاورت می‌توان ترتیب راس‌ها را هم عوض کرد، مثلا اگر ماتریسی 5 راس دادشته باشد می‌توان نوشت.

```
p.adjacency_matrix(vertices=[2,4,1,3,0])
```

درجه متوسط برای گراف	G.average_degree()
میانگین فاصله برای گراف	G.average_distance()
چگالی گراف	G.density()
قطر گراف	G.diameter()
درجه هر راس(برای مثال راس 0)	G.degree(0)
درجه تمام رئوس	G.degree()
درجه راس‌های دلخواه	G.degree(vertices=[0,1],labels=True)
دنباله درجات را به صورت کاهشی نشان می دهد	G.degree_sequence()

گراف مکمل

```
Y=Graph({0:[1,2],2:[3],4:[0,1]})
S=Y.complement()
S.plot()
```

رسم گراف کامل

```
graphs.CompleteGraph(5)
```

گراف کامل دو بخشی

```
K=graphs.CompleteBipartiteGraph(9,3)
```

ازراس x به کدام راس‌ها میتوان رفت.

```
G=Graph({0:[1,3],1:[2,3],2:[3],4:[5,6],5:[6],7:[8]})
G.connected_component_containing_vertex(0)
```

لیست گراف‌های جدا ازهم G.connected_components()

تعداد گراف‌های جدا ازهم G.connected_components_number()

دورهای پایه موجود درگراف G.cycle_basis()

برای نشان دادن مسیر اتصال دورها G.cycle_basis(output='edge')

دورهای موجود برای گراف جهت دار استفاده **نمی‌شود.**

پاک کردن یال

```
G=Graph([(0,2,'a'),(0,2,'b'),(0,1,'c'),(1,2,'d')],multiedges=True)
G.delete_edges([(1,2,'d'),(0,1,'c')])
```

که d وc برچسب یال است.

پاک کردن یال تکراری

```
G.delete_multiedge(0,2)
```

پاک کردن راس

```
G.delete_vertices([0,1])
```

دو جفت راس (x,y)و (z,w)داریم آیا مسیرهایی هست که راس xرا به y و z را به w وصل کند و این دو مسیر تداخل نداشته باشد؟

```
A=graphs.GridGraph([5,5])

P1,p2=A.disjoint_routed_paths([((0,0),(0,4)),((4,4),(4,0))])

P1.plot()

P2.plot()
```

گراف وزن دار

```
G=Graph({0:{1:3},1:{2:1},2:{3:1},3:{4:6},4:{0:2}})
```

{1:3}:0به معنی این است که راس 0 به راس 1 متصل است با وزن 3

```
G.plot(edge_labels=True).show()
```

فاصله بین دوتا راس با احتساب وزن یال‌ها

```
G.distance(0,3,by_weight=True)
```

مسیرهای بین دوتا راس با یال‌های مجزا (یعنی مسیرهایی که هیچ دو مسیری یال تکراری نداشته باشند.)

```
W=graphs.DodecahedralGraph()

W.edge_disjoint_paths(3,15)
```

مسیرهای بین دوتا راس با راس‌های مجزا

```
W.vertex_disjoint_paths(3,15)
```

دور اویلری: دور زدن در گراف بدون یال تکراری

```
G.eulerian_circuit()
```

دور همیلتونی: دور زدن در گراف بدون راس تکراری

```
G.is_hamiltonian()
G.hamiltonian_cycle()
```

اگر بخواهیم راسهای دور همیلتونی را نشان دهد از دستور زیر استفاده می کنیم.

```
G.hamiltonian_cycle(algorithm='backtrack')
```