Introduction to Graphs and Social Networks

Thomas Breuel

UniKL

Social Networks

people and their relations

- actors: people, organizations, groups, pages, products, ...
- relationships friendship, ownership, hug, poke, kinship, ...

online social networks

Examples: Google+, Facebook, Orkut, ...

Components:

- users
- relations
- profiles
- timelines
- pages/interests

traditional social networks

Examples:

- intermarriage among Venitian merchant families
- intermarriage in royal families
- message exchanges between employees of a corporation
- report structure in a corporation
- diplomatic relations between countries

citation analysis

Examples: bibliometrics, page rank, ...

Components:

- publications, books, web pages, ...
- citations, hyperlinks, ...

representations

- graphs
- directed graphs
- hypergraphs
- matrices
- Markov chains

datasets

Stanford Large Network Dataset Collection

- Social networks : online social networks, edges represent interactions between people
- Networks with ground-truth communities : ground-truth network communities in social and information networks
- Communication networks : email communication networks with edges representing communication
- Citation networks : nodes represent papers, edges represent citations
- Collaboration networks : nodes represent scientists, edges represent collaborations (co-authoring a paper)
- ...

http://snap.stanford.edu/data/

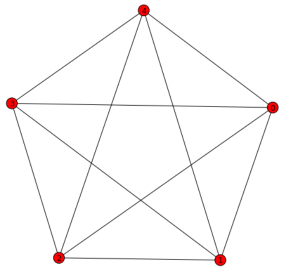
Review of Graphs

graphs

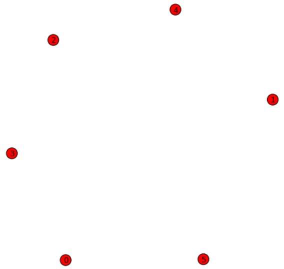
Graph: a set of vertices/nodes V and a set of edges/links $E \subseteq V \times V$

- directed vs undirected graph
- self loops or no self loops
- nodes have degrees, in/out-degrees

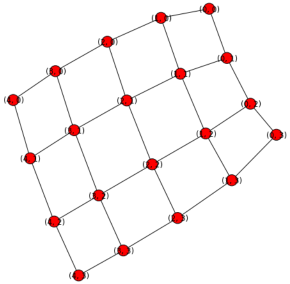
```
G = nx.complete_graph(5)
nx.draw(G)
```



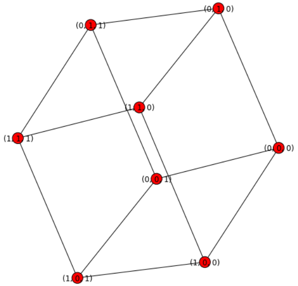
```
G = nx.empty_graph(6)
nx.draw(G)
```



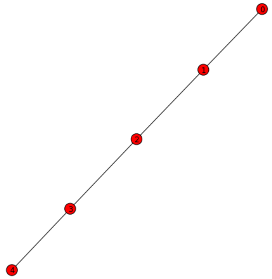
```
G = nx.grid_2d_graph(5,4)
nx.draw(G)
```



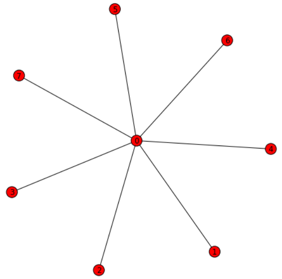
```
G = nx.hypercube_graph(3)
nx.draw(G)
```



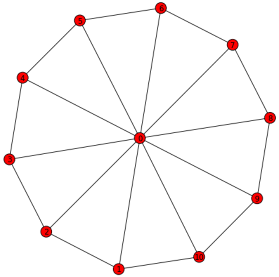
```
G = nx.path_graph(5)
nx.draw(G)
```



```
G = nx.star_graph(7)
nx.draw(G)
```

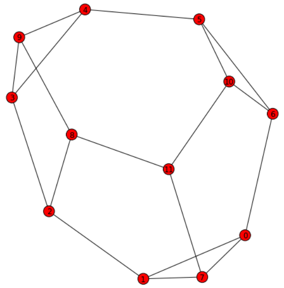


```
G = nx.wheel_graph(11)
nx.draw(G)
```

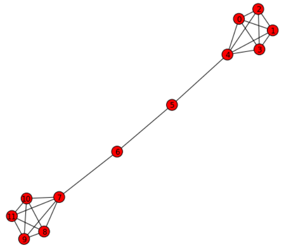


Frucht graph = cubic graph with only a single (trivial) automorphism

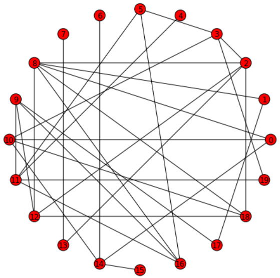
```
G = nx.frucht_graph()
nx.draw(G)
```



```python
G = nx.barbell_graph(5,2)
nx.draw(G)
```



```
G = nx.erdos_renyi_graph(20,0.2)
nx.draw_circular(G)
```



walks, trails, paths

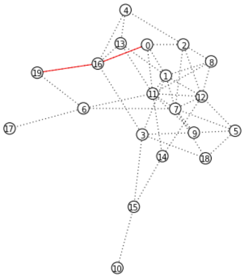
- a walk is a sequence of vertices and edges
- a trail is a walk in which all the edges are distinct
- a path is a walk in which all the edges and all the vertices are distinct
- a closed walk is a walk that starts and ends on the same vertex
- a cycle is a closed walk of at least three nodes, all edges are distinct, and only the start and end are the same
- a tour is a closed walk in which each edge is used at least once

distance in graphs

Graphs and paths define a kind of distance.

- a *geodesic* is the shortest path between two nodes
- the *distance* or *geodesic distance* of two nodes is the length of the shortest path between them
- the *diameter* of a graph is the longest distance in that graph
- the *eccentricity* of a node is the largest distance of that node to any other node

```
G = nx.erdos_renyi_graph(20,0.2)
p = nx.shortest_path(G,0,19)
pos = nx.spring_layout(G)
nx.draw_networkx_nodes(G,pos,node_color='w',node_size=200)
nx.draw_networkx_labels(G,pos,font_size=10)
nx.draw_networkx_edges(G,pos,style='dotted')
nx.draw_networkx_edges(G,pos,edgelist=zip(p,p[1:]),edge_color='r')
axis("off"); None
```



```python
while not nx.is_connected(G): G = nx.erdos_renyi_graph(20,0.2)
nx.average_shortest_path_length(G)
```

```
2.1894736842105265
```

connectivity

- graphs are connected or disconnected
- nodes are partitioned into *connected components* (subgraphs whose nodes have a finite distance with, and an infinite between them)
- a node is a *cutpoint* if removing it increases the number of connected components
- an edge is a *bridge* if removing it increases the number of connected components
- a *cutset* of nodes is a set of nodes such that removing it increases the number of connected components
- the *node connectivity* of a connected graph is the minimum size of a cutset

applications of connectivity

Node connectivity is related to failure. E.g., the node connectivity of a computer network (as a graph) tells you the minimum number of failures before the network becomes disconnected.

Connectivity may also be related to group structure of a social network, identifying weak or attackable links, etc.

Continuous generalizations are based on flow and the effect of cuts on flow.

Concepts from flow and connectivity are also used in image segmentation.

```
G = nx.erdos_renyi_graph(20,0.3)
# nx.minimum_node_cut(G)
```

```
for node in range(20):
    for e in G[node]:
        G[node][e]["capacity"] = 1.0
nx.min_cut(G,0,1,"capacity")
```

```
6.0
```

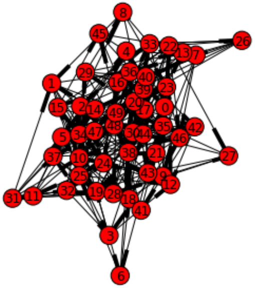
example: condensation

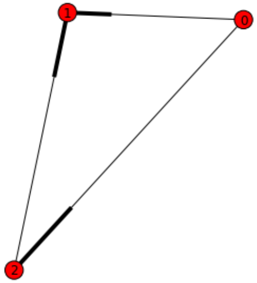
This is mainly one of many examples of fun and potentially useful algorithms and concepts lurking in graph libraries. Browse and experiment!

A *condensation* of a directed graph is a directed graph in which all the strongly connected components have been turned into a single node.

A *strongly connected component* of a directed graph is a connected component in which all nodes are reachable.

```
G = nx.fast_gnp_random_graph(50,0.1,3,1)
GC = nx.condensation(G)
subplot(121); nx.draw(G)
subplot(122); nx.draw(GC)
```





isomorphism and subgraphs

All graphs described so far are *unlabeled*: they have no labels distinguishing nodes or edges from one another and their entire structure is determined by their connectivity.

A *graph isomorphism* from $G_1 = (V_1, E_1)$ to $G_2 = (V_2, E_2)$ is a mapping $f : V_1 \to V_2$ such that $(f(u), f(v)) \in E_2$ iff $(u, v) \in E_1$.

A *subgraph* is a graph consisting of subset of vertices and edges of another graph.

Maximum subgraph isomorphism is the problem of finding maximum subgraphs of G_1 and G_2 such that the subgraphs are isomorphic.

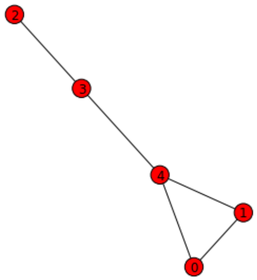
Question: What is the complexity of graph isomorphism? What is the complexity of subgraph ismomorphism? How would you solve these problems?

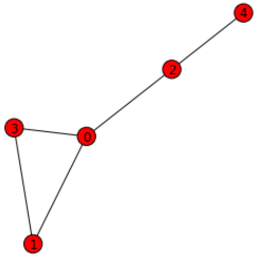
isomorphism

```
G1 = nx.erdos_renyi_graph(5,0.4)
while 1:
    G2 = nx.erdos_renyi_graph(5,0.4)
    GM = nx.isomorphism.GraphMatcher(G1,G2)
    if GM.is_isomorphic(): break
print GM.mapping
```

```
{0: 1, 1: 3, 2: 4, 3: 2, 4: 0}
```

```
subplot(121); nx.draw(G1)
subplot(122); nx.draw(G2)
```



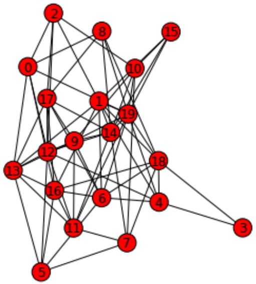


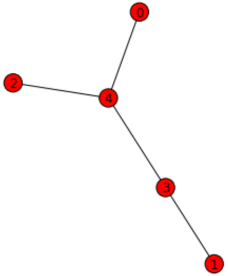
subgraph isomorphism

```python
while 1:
    G1 = nx.erdos_renyi_graph(20,0.3)
    if nx.is_connected(G1): break
while 1:
    G2 = nx.erdos_renyi_graph(5,0.3)
    if not nx.is_connected(G2): continue
    GM = nx.isomorphism.GraphMatcher(G1,G2)
    if GM.subgraph_is_isomorphic(): break
print GM.mapping
```

```
{0: 0, 1: 2, 2: 4, 10: 3, 7: 1}
```

```
subplot(121); nx.draw(G1)
subplot(122); nx.draw(G2)
```





special kinds of graphs

The *complement* of a graph $G = (V, E)$ is a graph $\bar{G} = (V, \bar{E})$ such that $(u, v) \in \bar{E}$ iff $(u, v) \notin E$.

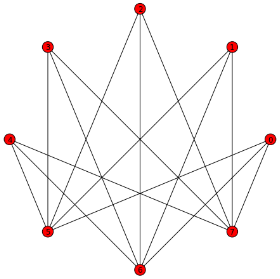
A *tree* is an acyclic, connected graph.

A tree is *minimally connected* since every edge is a bridge.

A *forest* is an acyclic graph.

A *bipartite graph* is a graph $G = (V_1 \cup V_2, E)$ where all the edges are of the form (u, v) with $u \in V_1$ and $v \in V_2$.

```
G = nx.complete_bipartite_graph(5,3)
nx.draw_circular(G)
```



cliques

A *clique* is a maximally connected subgraph.

A *maximal clique* is a clique that is not a subgraph of a larger clique.

A *maximum clique* is the largest clique in a graph.

Finding a maximum clique is NP-hard.

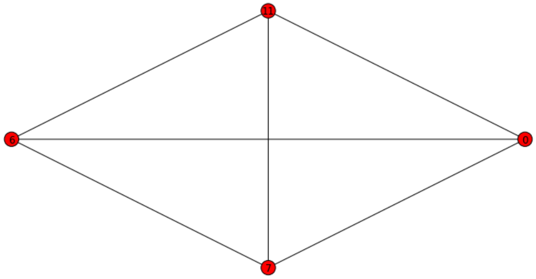
An *independent set* is a clique in the complement of a graph.

```python
G = nx.erdos_renyi_graph(20,0.4)
mc = nx.graph_clique_number(G)
print mc
```

```
4
```

```python
for C in nx.find_cliques(G):
     if len(C)==mc: break
print C
C = G.subgraph(C)
nx.draw_circular(C)
```

```
[0, 11, 6, 7]
```



Centrality and Prestige

reminder

What we are doing here and below is to try to identify mathematical descriptors, indexes, numerical measures based on graphs that...

- correspond to some intuitive concept of agents and their relations OR
- correlate with some other concept or numerical measure we are interested in

E.g., are highly connected individuals on social networks happier than those who are less connected?

centrality

Often, we ask the question: who are the “most important” actors or “most central” actors in a social network (Facebook, criminal network, whatever.)

- most connected / most number of friends
- closest to the rest of the network, lowest average/maximum geodesic distance
- on most geodesic paths between other actors (nodes the NSA likes)
- frequency with which node is on random walks connecting two other nodes

```python
G = nx.Graph()
with os.popen("zcat␣Data/facebook_combined.txt.gz") as stream:
    for line in stream.readlines():
        u,v = [int(x) for x in line[:-1].split()]
        G.add_edge(u,v)
N = G.number_of_nodes()
print N
```

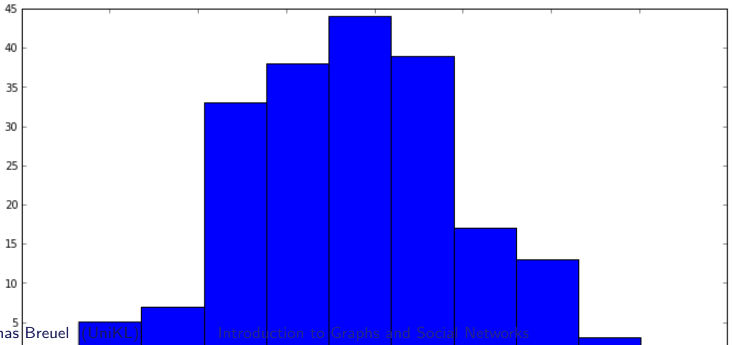
```
4039
```

```
N = 200
G = nx.erdos_renyi_graph(N,0.2)
```

The degree centrality for a node v is the fraction of nodes it is connected to.

```
c = nx.degree_centrality(G)
c = array([c[i] for i in range(N)])
_=hist(log10(c))
print find(c==amax(c)),amax(c)
```

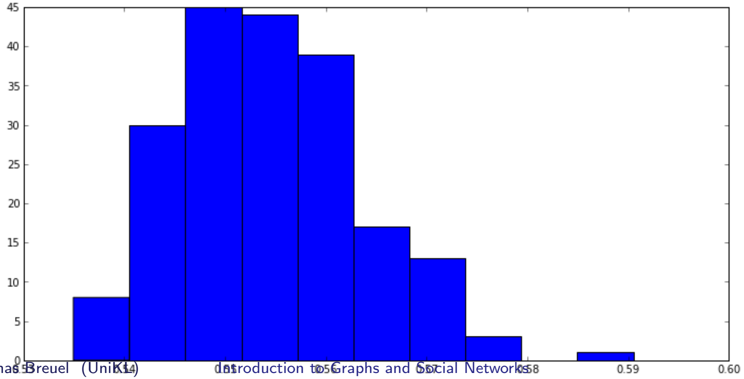
```
[128] 0.306532663317
```



Closeness centrality at a node is 1/average distance to all other nodes.

```
c = nx.closeness_centrality(G)
c = array([c[i] for i in range(N)])
_ = hist(c)
print find(c==amax(c))
```

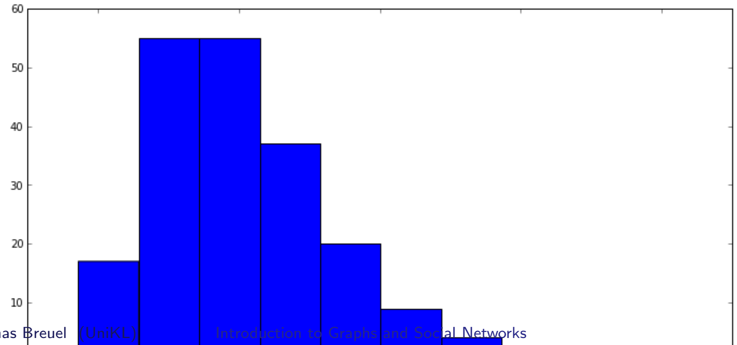
```
[128]
```



Betweenness centrality of a node v is the sum of the fraction of all-pairs shortest paths that pass through v

```
c = nx.betweenness_centrality(G)
c = array([c[i] for i in range(N)])
_ = hist(c)
print find(c==amax(c))
```

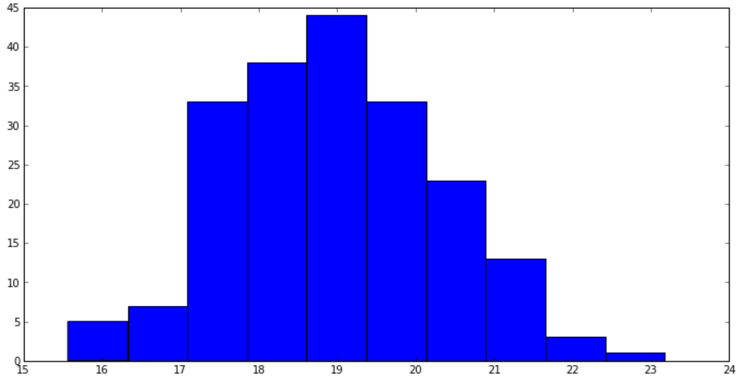
```
[128]
```



Compute current-flow closeness centrality for nodes. A variant of closeness centrality based on effective resistance between nodes in a network. This metric is also known as information centrality.

```
c = nx.current_flow_closeness_centrality(G)
c = array([c[i] for i in range(N)])
_ = hist(c)
print find(c==amax(c))
```

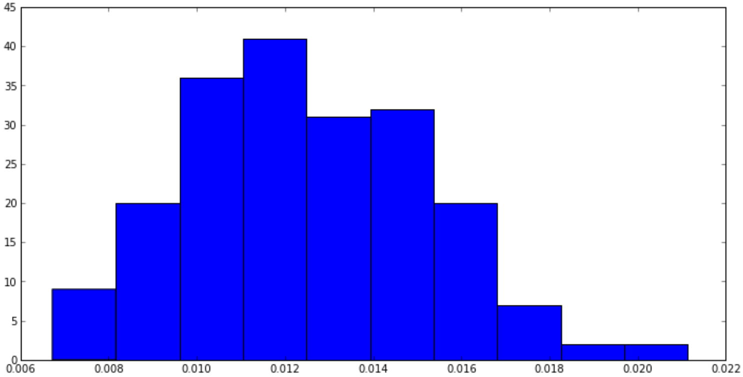
```
[128]
```



Current-flow betweenness centrality uses an electrical current model for information spreading in contrast to betweenness centrality which uses shortest paths. Current-flow betweenness centrality is also known as random-walk betweenness centrality.

```
c = nx.approximate_current_flow_betweenness_centrality(G)
c = array([c[i] for i in range(N)])
_ = hist(c)
print find(c==amax(c))
```

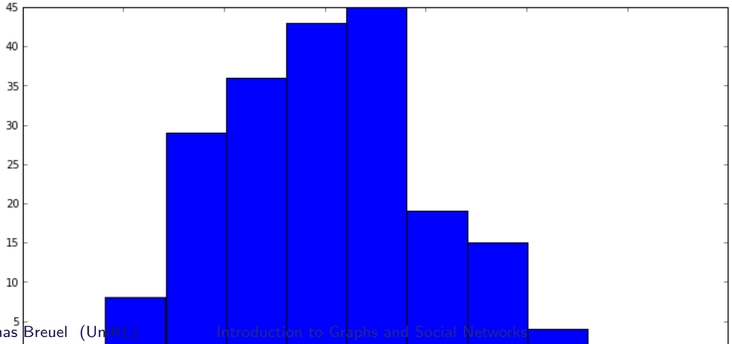
```
[128]
```



Uses the power method to find the eigenvector for the largest eigenvalue of the adjacency matrix of G. This is related to Google's page rank.

```python
c = nx.eigenvector_centrality(G)
c = array([c[i] for i in range(N)])
_ = hist(c)
print find(c==amax(c))
```

```
[128]
```



uses of centrality

What can we do with these measures?

What do these centrality measures correlate with? Income? Happiness? Level in a corporate hierarchy? Leadership in a criminal organization? Etc.

Do some centrality measures correlate better with these variables than others?

prestige

In addition to centrality, there are also measures for directed graphs, like *prestige*.

E.g., if your paper is cited by someone else, it indicates higher prestige for your paper.

You citing someone else's paper does not increase the prestige of your paper.

Like centrality, there are various different prestige measures.

Structural Balance

balance

Assume that relations among actors can either be positive (friends) or negative (foes).

Some patterns of relationships are more “balanced” than others because they are particularly consistent or transitive.

If A is friends with B and B is friends with C, we expect A to be friends with C as well. If they are not, then structural balance is violated.

balance

Definitions of Balance:

- A cycle is balanced if the product of its edges is positive.
- A signed graph is balanced iff all cycles have positive signs.

Numerical indexes:

- The *index of balance* of a signed graph is the fraction of unbalanced cycles among all cycles.
- The *line index of balance* of a signed graph is the fraction of edge signs that need to be changed to make it balanced.

clustering based on balance

A balanced signed graph can be divided into sets of nodes that only have positive relations within each group.

Cohesive Subgroups

subgroups

Tightly connected groups of people are another concept that's of interest in social network analysis.

They often indicate a group of people holding very similar beliefs or cooperating closely.

The simplest tightly connected subgroup is a maximal clique; we already talked about these.

n-cliques

Instead of requiring a subgraph in which every node is connected to every other, we can require a small geodesic distance.

An n *-clique* is a maximal subgraph such that the largest geodesic distance between any two nodes is no greater than n .

$n = 1$ reduces to the usual clique.

$n = 2$ is commonly used: it's robust and simple, but not too large.

n -cliques are not all that useful; e.g., geodesics may include nodes outside the clique.

n-clans and n-clubs

An n -clan is an n -clique with a diameter less than or equal to n .

An n -club is a subgraph of diameter less than or equal to n .

A k -core is a subgraph in which each node is adjacent to at least k nodes within the subgraph.

Clustering Coefficient

clustering coefficient

The *clustering coefficient* of a graph is a measure of the degree to which the graph's nodes tend to cluster together.

- a *triplet* is three nodes connected by two or three edges
- a *triangle* is three closed triplets (one centered on each node)

The clustering coefficient is:

$$C = \frac{\text{number of closed triplets}}{\text{number of triplets}} = \frac{3 \times \text{number of triangles}}{\text{number of triplets}}$$

```
G = nx.erdos_renyi_graph(100,0.2)
nx.average_clustering(G)
```

```
0.2108177882820873
```

Other Graphs

multigraphs and hypgergraphs

In graphs with *labeled edges*, edges are triples $(u, v, l) \in V \times V \times L$ for some set of labels L .

In *multigraphs* (labeled or not), multiple edges may occur between two vertices.

In *hypergraphs*, edges are subsets of V^d for some degree or set of degrees d .

relations

In mathematics, we have a closely related concept of *relations*. Examples of relations are inequality and equality $=$.

The relations of a set V are a subset of $V \times V$. This is the same as a directed graph.

Relations are often defined on infinite sets, but graphs are defined on finite sets of vertices.

Common terminology:

- *reflexive*: $u \triangleleft u$
- *symmetric*: $u \triangleleft v \Rightarrow v \triangleleft u$
- *transitive*: $u \triangleleft v \wedge v \triangleleft w \Rightarrow u \triangleleft w$

Matrices

adjacency matrix

We often represent graphs as adjacency matrices.

Given a graph (V, E) ,

$$A_{ij} = \lfloor (i, j) \in E \rfloor$$

A generalization of a graph is a *weighted graph* in which edges are associated with numbers. They have adjacency matrices with more general values than zero or one.

adjacency matrices and reachability

Adjacency matrices are closely related to reachability, random walks, Markov chains.

Given:

$$v = (0, ..., 1, ...0)$$

Successor states:

$$v' = A \cdot v$$