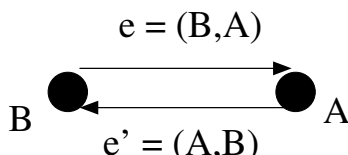


Directed Graphs

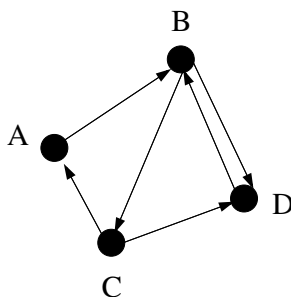
So far we have been working with graphs with undirected edges. Graphs with directed edges are also very useful in computer science. A directed edge is an edge where the endpoints are distinguished - one is the head and one is the tail. Thus its endpoints form an *ordered pair*.



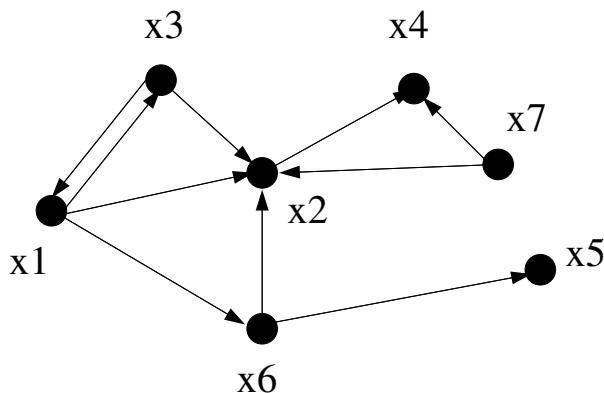
A graph with directed edges is called a *directed graph*, or *digraph*. Such graphs can have directed edges in both directions:



Note that we treat e and e' as different edges in the graph above. With directed graphs, the notion of degree splits into *indegree* and *outdegree*. In the following digraph $\text{indegree}(D) = 2$ while $\text{outdegree}(D) = 1$.



Directed graphs arise in all sorts of applications. One interesting example is the digraph that represents the hyperlink structure of the World Wide Web. In this example, every page or file is represented by a node and there is a directed edge from node x to node y if the page associated with x has a link to the page associated with y . For example, in the following graph the vertices $V = \{x_1, \dots, x_N\}$ are web pages and $x_i \rightarrow x_j$ if x_i contains a hyperlink to x_j .



The web graph is an enormous graph with many billions and probably even trillions of nodes. At first glance, this graph wouldn't seem to be very interesting. But in 1995, two students at Stanford, Larry Page and Sergey Brin realized that the structure of this graph could be very useful in building a search engine. Traditional document searching programs had been around for a long time and they worked in a fairly straightforward way. Basically, you would enter some search terms and the searching program would return all documents containing those terms. A relevance score might also be returned for each document based on the frequency or position that the search terms appeared in the document. For example, if the search term appeared in the title or appeared 100 times in a document, that document would get a higher score. So if an author wanted a document to get a higher score for certain keywords, he would put the keywords in the title and make it appear in lots of places. You can even see this today with some bogus web sites.

This approach works fine if you only have a few documents that match a search term. But on the web, there are billions of documents and millions of matches to a typical search. For example, doing a search on Google for “math for computer science notes” gives over 4 million hits! How does Google decide which 10 or 20 to show first? It wouldn't be smart to pick a page that gets a high keyword score just because it has “math math . . . math” across the front of the document.

One way to get placed high on the list is to pay Google an advertising fee: Google gets an enormous revenue stream from these fees. Of course an early listing is worth a fee only if an advertiser's target audience is attracted to the listing. But an audience does get attracted to Google listings because its ranking method is really good at determining the most important relevant web pages. For example, Google demonstrated its accuracy in our case by giving fourth rank to the Spring 2005 open courseware page for 6.042 :-) while Google did not even know I was interested in MIT. So how did Google know to pick 6.042 to be fourth out of 4 million?

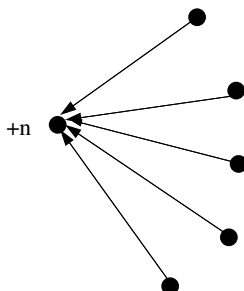
Well back in 1995, Larry and Sergey got the idea to allow the digraph structure of the web to determine which pages are likely to be the most important. Then they would show those first in response to a search request.

1 A First Crack at Page Rank

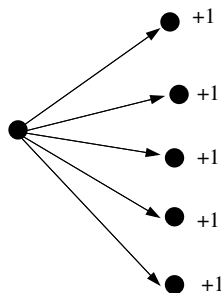
Looking at the web graph, any idea which node/page might be the best to rank 1st? Assume that all the pages match the search terms for now. Well, intuitively, we should choose x_2 since lots of other pages point to it. This leads us to their first idea.

Define the page rank of x to equal $\text{indegree}(x)$, which we denote as $PR(x)$. Then, of the pages that match the search terms, list those with the highest page rank first. The idea is to think of web pages as voting for the most important page - the more votes, the better the rank.

Of course, there are some problems with this idea. Suppose you want your page to have a high ranking. One thing you could do is to create lots of dummy pages with links to your page.



There is another problem —a page could become unfairly influential by having lots of links to other pages, thereby increasing the rank of lots of other pages.



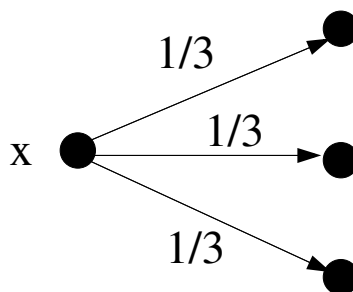
So this strategy for high ranking would amount to, “vote early, vote often,” which is no good if you want to build a search engine that’s worth paying fees for. So, admittedly, their original idea was not so great. It was better than nothing, but certainly not worth billions of dollars.

2 Random Walk on the Web Graph

But then Sergey and Larry thought some more and came up with a couple of improvements. Instead of just counting the indegree of a node, they considered the fraction of time of being

at each page after a long random walk on the web graph. In particular, they decided to model a user's web experience as following each link on a page without preferring one link above another link.

They decided to give each page exactly one vote: a given page divides its vote without preference equally among its outgoing links. So, if a node in the web graph has outdegree d , then every link gets $1/d$ of the vote. In general, an edge has weight $1/\text{outdegree}(x)$ if its tail is x .



Now any page has the same impact regardless of the number of links it has. The user experience is just a random walk on the web graph (this will become clear when we will cover the basics of probability theory).

Sergey and Larry decided to compute the expected fraction of time that a user spends at each page and to let these fractions determine the pageranks. The hope is that dummy pages will have low page rank and thus won't influence the average rank of real pages.

How do we compute the expected fraction of time that a user spends at each page? Redefine the pagerank $PR(x)$ as the fraction of time spend in node x . A user may enter node x over one of its incoming edges $y \rightarrow x$. Since the user can choose any of the outgoing edges from y with equal vote and since the user spends an expected fraction of time $PR(y)$ in y , this contributes $PR(y)/\text{outdegree}(y)$ to the fraction of time spend in x . Summing over all possible incoming edges gives

$$\forall x, PR(x) = \sum_{y \text{ s.t. } y \rightarrow x} \frac{PR(y)}{\text{outdegree}(y)}.$$

For example, in our web graph, we have

$$PR(x_4) = \frac{PR(x_7)}{2} + \frac{PR(x_2)}{1}.$$

One can also think of this as x_7 sending $\frac{1}{2}$ of its page rank to x_2 and the other half to x_4 ; x_2 sends all of its page rank to x_4 .

This provides a set of n linear equations in n variables where n is the number of nodes, or pages, and the variables are the page ranks. Since the page rank of x should represent

the fraction of time spend in node x , the page ranks should sum up to 1. This gives one additional equation

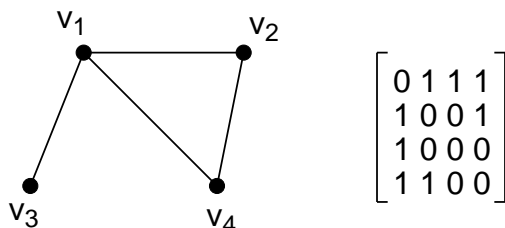
$$\sum_x PR(x) = 1.$$

Now, the set of linear equations has one more equation than the number of variables: it's not even clear that there is a solution to this system of equations.

But Sergey and Larry were smart fellows and they set up their page rank algorithm so it would always have a meaningful solution that is non-negative and satisfies the extra equation which states that the sum of the page ranks is equal to 1. (It turns out that the page rank computed for a page is the probability that a surfer lands on that page after a long random walk through the web starting from a random page and clicking on links uniformly. This will become clear when we will cover the basics of probability theory.) To solve the system of equations to compute the page ranks, it is helpful to represent the web graph by its adjacency matrix.

3 Adjacency Matrices

A graph can be represented by an *adjacency matrix*. In particular, if a graph has vertices v_1, \dots, v_n , then the adjacency matrix is $n \times n$. The entry in row i , column j is 1 if there is an edge $v_i \rightarrow v_j$ and is 0 if there is no such edge. For example, here is a graph and its adjacency matrix:



The adjacency matrix of an undirected graph is always symmetric about the diagonal line running from the upper left entry to the lower right. The adjacency matrix of a directed graph need not be symmetric, however. Entries on the diagonal of an adjacency matrix are nonzero only if the graph contains self-loops.

Adjacency matrices are useful for two reasons. First, they provide one way to represent a graph in computer memory. Second, by mapping graphs to the world of matrices, one can bring all the machinery of linear algebra to bear on the study of graphs. For example, one can analyze a highly-prized quality of graphs called “expansion” by looking at the so-called eigenvalues of the adjacency matrix. (In a graph with good expansion, the number of edges departing each subset of vertices is at least proportional to the size of the subset. This is not so easy to achieve when the graph as a whole has few edges, say $|E| = 3|V|$.) Here we

prove a simpler theorem in this vein. If M is a matrix, then $M_{i,j}$ denotes the entry in row i , column j . Let M^k denote the k -th power of M . As a special case, M^0 is the identity matrix I . This is the matrix with ones on its diagonal and zeroes elsewhere.

Theorem 1. *Let G be a digraph (possibly with self-loops) with vertices v_1, \dots, v_n . Let M be the adjacency matrix of G . Then $M_{i,j}^k$ is equal to the number of length- k walks from v_i to v_j .*

Proof. We use induction on k . The induction hypothesis is that $M_{i,j}^k$ is equal to the number of length- k walks from v_i to v_j , for all i, j .

Each vertex has a length-0 walk only to itself. Since $M_{i,j}^0 = 1$ if and only if $i = j$, the hypothesis holds for $k = 0$.

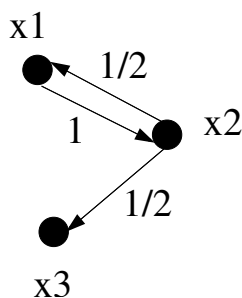
Now suppose that the hypothesis holds for some $k \geq 0$. We prove that it also holds for $k + 1$. Every length- $(k + 1)$ walk from v_i to v_j consists of a length k walk from v_i to some intermediate vertex v_m followed by an edge $v_m \rightarrow v_j$. Thus, the number of length- $(k + 1)$ walks from v_i to v_j is equal to:

$$M_{i,1}^k M_{1,j} + M_{i,2}^k M_{2,j} + \dots + M_{i,n}^k M_{n,j}$$

This is precisely the value of $M_{i,j}^{k+1}$, so the hypothesis holds for $k + 1$ as well. The theorem follows by induction. \square

4 Web Graph weighted by Page Ranks

For a *weighted* graph, the weighted adjacency matrix is W , where $W_{i,j}$ is the weight on edge $i \rightarrow j$ if $(i, j) \in E$, and is 0 otherwise. For example, the graph



would have the weighted adjacency matrix

$$\begin{pmatrix} 0 & 1 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 0 \end{pmatrix}$$

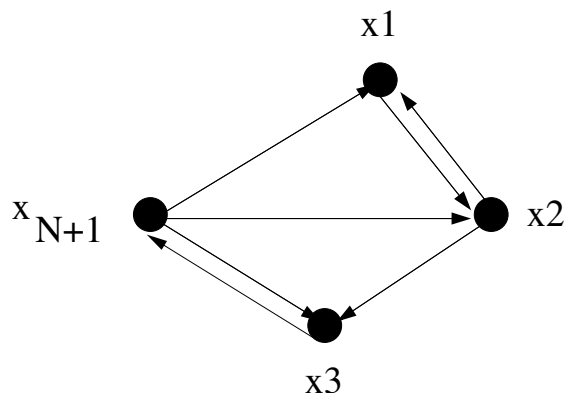
Now for Sergey and Larry's graph, the edge weights are just $1/\text{outdegree}(x)$ for nodes x . The row sums in such a weighted graph must equal 0 or 1 since in the x th row, each

non-zero entry, if any, has weight $1/\text{outdegree}(x)$. Of course some rows may have zero sum, and these correspond to pages with no hyperlinks out, i.e., nodes with outdegree 0. These nodes are called *sinks* in a digraph because once you are there, you can't get out.

Definition. A sink is a node with outdegree 0.

There's one aspect of the web graph described thus far that doesn't mesh with the user experience: some pages have no hyperlinks out. Under the current model, the user cannot escape these pages. In reality, however, the user doesn't fall off the end of the web into a void of nothingness. Instead, he restarts his web journey.

To model this aspect of the web, Sergey and Larry added a supernode to the web graph and had every page with no hyperlinks point to it. Moreover, the supernode points to every other node in the graph, allowing you to restart the walk from a random place.



Now you can't get stuck at any node. The addition of the supernode also removes the possibility that the value $1/\text{outdegree}(x)$ might involve a division by zero. The weighted adjacency matrix W' now is 4×4 and looks like

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 \end{pmatrix}$$

Now all the row sums are one. This turns out to make the page rank calculation better. Sergey and Larry also did a lot of other stuff, but we don't have time to go into it here, and also most of it is closely held secret!

The nice thing about the weighted adjacency matrix is that it gives a very simple form for expressing the linear equations for page rank. If you put the page ranks in a vector \vec{P} :

$$\begin{pmatrix} PR(x_1) \\ PR(x_2) \\ \dots \\ PR(x_N) \end{pmatrix}$$

Then $W^T \vec{P} = \vec{P}$, where W^T denotes the transpose of W . Note that this follows from the fact that the equations have the form $\sum_{1 \leq i \leq n} W_{i,j} PR(x_i) = PR(x_j)$ for each $1 \leq j \leq n$.

If you have taken a linear algebra or numerical analysis course, you realize that the vector of page ranks is just the principle eigenvector of the weighted adjacency matrix of the web graph! Once you've had such a course, these values are easy to compute. Of course, when you are dealing with matrices of this size, the problem gets a little more interesting. Just keeping track of the digraph whose nodes are billions of web pages is a daunting task. That's why Google is building power plants. Indeed, Larry and Sergey named their system Google after the number 10^{100} , called "googol", to reflect the fact that the web graph is so enormous.

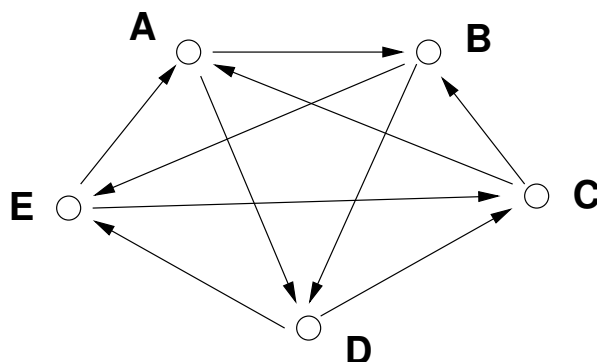
Anyway, now you can see how 6.042 ranked fourth out of 4 million matches. Lots of other universities use our notes and probably have links to the 6.042 open courseware site and they are themselves legitimate, which ultimately leads 6.042 to get a high page rank in the web graph.

The moral of this story is that you too can become a billionaire if you study your graph theory!

5 Tournament Rankings

Suppose that n players compete in a round-robin tournament. Thus, for every pair of players u and v , either u beats v or else v beats u . Interpreting the results of a round-robin tournament can be problematic. There might be all sorts of cycles where x beat y , y beat z , yet z beat x . Graph theory provides at least a partial solution to this problem.

The results of a round-robin tournament can be represented with a *tournament graph*. This is a directed graph in which the vertices represent players and the edges indicate the outcomes of games. In particular, an edge from u to v indicates that player u defeated player v . In a round-robin tournament, every pair of players has a match. Thus, in a tournament graph there is either an edge from u to v or an edge from v to u for *every* pair of vertices u and v . Here is an example of a tournament graph:



Its adjacency matrix is

$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \end{pmatrix}$$

(Notice that if M is the adjacency matrix of a tournament graph, then $I + M + M^T$, the addition of the identity matrix, the adjacency matrix, and its transpose, is equal to the all-one matrix.)

The notions of walks, Euler tours, and Hamiltonian cycles all carry over naturally to directed graphs. A **directed walk** is an alternating sequence of vertices and directed edges:

$$v_0, v_0 \longrightarrow v_1, v_1, v_1 \longrightarrow v_2, v_2, \dots, v_{n-1}, v_{n-1} \longrightarrow v_n, v_n$$

A **directed Hamiltonian path** is a directed walk that visits every vertex exactly once.

We're going to prove that in every round-robin tournament, there exists a ranking of the players such that each player lost to the player ranked one position higher. For example, in the tournament above, the ranking

$$A > B > D > E > C$$

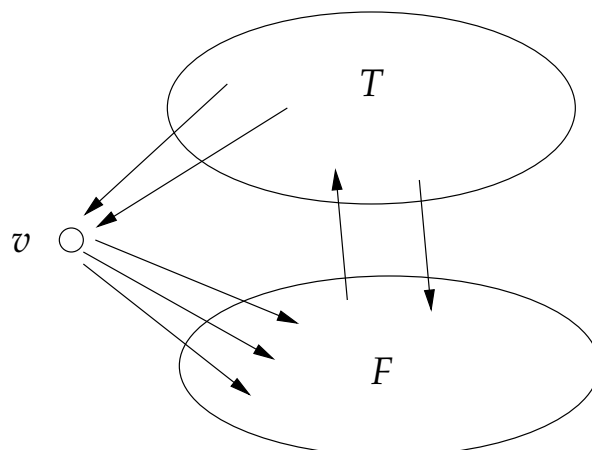
satisfies this criterion, because B lost to A , D lost to B , E lost to D and C lost to E . In graph terms, proving the existence of such a ranking amounts to proving that every tournament graph has a Hamiltonian path.

Theorem 2. *Every tournament graph contains a directed Hamiltonian path.*

Proof. We use strong induction. Let $P(n)$ be the proposition that every tournament graph with n vertices contains a directed Hamiltonian path.

Base case. $P(1)$ is trivially true; every graph with a single vertex has a Hamiltonian path consisting of only that vertex.

Inductive step. For $n \geq 1$, we assume that $P(1), \dots, P(n)$ are all true and prove $P(n+1)$. Consider a tournament with $n+1$ players. Select one vertex v arbitrarily. Every other vertex in the tournament either has an edge *to* vertex v or an edge *from* vertex v . Thus, we can partition the remaining vertices into two corresponding sets, T and F , each containing at most n vertices.



The vertices in T together with the edges that join them form a smaller tournament. Thus, by strong induction, there is a Hamiltonian path within T . Similarly, there is a Hamiltonian path within the tournament on the vertices in F . Joining the path in T to the vertex v followed by the path in F gives a Hamiltonian path through the whole tournament. (As special cases, if T or F is empty, then so is the corresponding portion of the path.) \square

The ranking defined by a Hamiltonian path is not entirely satisfactory. In the example tournament, notice that the lowest-ranked player (C) actually defeated the highest-ranked player (A)!

Maybe there is another way to analyze tournaments. As the following section explains, there is always a “best” player who is able to defeat any given player or some other player who defeats the given player.

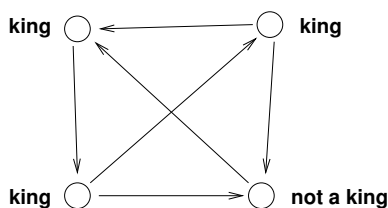
6 The King Chicken Theorem

Consider the following situation. There are n chickens in a farmyard. For each pair of distinct chickens, either the first pecks the second or the second pecks the first, but not both. We say that chicken u *virtually pecks* chicken v if either:

- Chicken u pecks chicken v .
- Chicken u pecks some other chicken w who in turn pecks chicken v .

A chicken that virtually pecks every other chicken is called a *king chicken*.

We can model this situation with a tournament digraph. The vertices are chickens, and an edge $u \rightarrow v$ indicates that chicken u pecks chicken v . In the tournament below, three of the four chickens are kings.



Now we're going to prove that a chicken tournament always results in at least one chicken king.

Theorem 3 (King Chicken Theorem). *The chicken with highest outdegree in an n -chicken tournament with $n \geq 1$ is king.*

Proof. (By contradiction.) Let u be the node in the tournament graph with highest outdegree and suppose that u is not king. Let V be the set of nodes to which u has directed edges. Then, the number of nodes in V is equal to the outdegree of u : $v \in V$ iff $u \rightarrow v$. Since u is not king, there exists a node x such that

- there is not a directed edge from u to x , and
- for all nodes v with a directed edge $u \rightarrow v$ (that is, $v \in V$), there is not a directed edge from v to x .

Since in a tournament there exists exactly one directed edge between any two nodes, these two conditions are equivalent to

- there is a directed edge from x to u , and
- for all nodes $v \in V$, there is a directed edge from x to v .

In other words, the outdegree of x is at least one more than the number of nodes in V . So, the outdegree of x is at least one more than the outdegree of u . But u is a node with highest outdegree. Contradiction! \square