Review: Recurrence relations (Chapter 8)

Last time we started in on recurrence relations. In computer science, one of the primary reasons we look at solving a recurrence relation is because many algorithms, whether "really" recursive or not (in the sense of calling themselves over and over again) often are implemented by breaking the problem down into smaller parts and solving those. In order to get a solid sense of their runtime, we need to be able to either solve the relation or at least get a *sense* of its complexity (think big-O).

Recall the Towers of Hanoi problem (moving discs) and it's solution:

```
ToH(n,A,B,C) : move n discs from A to C using B if necessary.
    If n=1, move the disc from A to C and stop
    ToH(n-1,A,C,B) <-------------- from A to B using C
    Move disc n from A to C
    ToH(n-1,B,A,C) <-------------- from B to C using A
```

Define T(n) = number of moves when there are n discs.

- T(1) = 1
- T(n) = 2T(n-1) + 1

In class last time we guessed at the solution and used an inductive proof to show that the guess (that $T(n)=2^n-1$) was correct. Today we are going to look at how to find closed form solutions to these problems generally.

Last time we worked through solving "*linear, homogeneous, recurrence relations with constant coefficients*" of degree 2

Solving Linear Recurrence Relations (8.2)

A *linear homogeneous recurrence relation of degree k with constant coefficients* is a recurrence relation of the form

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k},$$

where $c_1, c_2, \ldots, c_k$ are real numbers, and $c_k \neq 0$.

- The recurrence is <u>*linear*</u> because the all the "a_n" terms are just the terms (not raised to some power nor are they part of some function). So $a_n=2a_{n-1}$ is linear but $a_n=2(a_{n-1})^2$ is not.
- It is <u>*homogeneous*</u> because *all* terms are multiples of some previous value of a_n. So $a_n=2a_{n-1}$ is homogeneous, but $a_n=2a_{n-1}+1$ is not.
- It is of <u>*degree*</u> k because a_n is expressed in terms of the last k terms of the sequence.
- And it has <u>*constant coefficients*</u> because all the c terms are constants (not a function of n).

We claimed (without proof) that finding the "characteristic equation" would help.

$$r^n = c_1 r^{n-1} + c_2 r^{n-2} + \cdots + c_k r^{n-k}$$

Now, if we divide both sides of this equation by r^{n-k} and move the things around a bit, we get:

$$r^k - c_1 r^{k-1} - c_2 r^{k-2} - \cdots - c_{k-1} r - c_k = 0.$$

Some new examples:

Indicate if the following are linear, homogeneous and have constant coefficients. If they are, find the characteristic equation associated with the recursion.

1. $a_n=2a_{n-1}+a_{n-2}$
2. $a_n=a_{n-1}+a_{n-2}+4$
3. $a_n=2a_{n-1}-4a_{n-2}$
4. $a_n=a_{n-2}+a_{n-3}$
5. $a_n=na_{n-1}+a_{n-2}$

Working with LH RR with CC and degree 2

Let c_1 and c_2 be real numbers. Suppose that $r^2 - c_1 r - c_2 = 0$ has two distinct roots r_1 and r_2. Then the sequence $\{a_n\}$ is a solution of the recurrence relation $a_n = c_1 a_{n-1} + c_2 a_{n-2}$ if and only if $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ for $n = 0, 1, 2, \ldots$, where α_1 and α_2 are constants.

That feels like it jumps out of nowhere. Let's focus on what it says, time allowing we will try to show it makes sense (or at least a bit of sense). The proof is on pages 515 and 516 of the text.

Consider the Fibonacci sequence defined by $a_n=a_{n-1}+a_{n-2}$. This is saying we need to find the roots of the characteristic equation and then the solution for this relation is of the form $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$, where r_1 and r_2 are those roots. Notice that α is just an arbitrary constant (that we'd have to figure out…)

Last time we worked out examples 3 and 4 from the text. Let's try this one: **$a_n=3a_{n-1}-2a_{n-2}$; $a_0=3$, $a_1=4$**

Repeated roots

Let c_1 and c_2 be real numbers with $c_2 \neq 0$. Suppose that $r^2 - c_1 r - c_2 = 0$ has only one root r_0. A sequence $\{a_n\}$ is a solution of the recurrence relation $a_n = c_1 a_{n-1} + c_2 a_{n-2}$ if and only if $a_n = \alpha_1 r_0^n + \alpha_2 n r_0^n$, for $n = 0, 1, 2, \ldots$, where α_1 and α_2 are constants.

Let's do $a_n=6a_{n-1} -9a_{n-2}$ where $a_0=1$ and $a_1=6$

- What is the characteristic equation?
- What are the roots of the characteristic equation?
- What is the solution to the recurrence?

General form for arbitrary degree

In this class, we will only be working with relations of this type of degree 2. But there is a more general result.

Let $c_1, c_2, \ldots, c_k$ be real numbers. Suppose that the characteristic equation

$$r^k - c_1 r^{k-1} - \cdots - c_k = 0$$

has t distinct roots $r_1, r_2, \ldots, r_t$ with multiplicities $m_1, m_2, \ldots, m_t$, respectively, so that $m_i \geq 1$ for $i = 1, 2, \ldots, t$ and $m_1 + m_2 + \cdots + m_t = k$. Then a sequence $\{a_n\}$ is a solution of the recurrence relation

$$a_n = c_1 a_{n-1} + c_2 a_{n-2} + \cdots + c_k a_{n-k}$$

if and only if

$$\begin{aligned} a_n = {} & (\alpha_{1,0} + \alpha_{1,1} n + \cdots + \alpha_{1,m_1-1} n^{m_1-1}) r_1^n \\ & + (\alpha_{2,0} + \alpha_{2,1} n + \cdots + \alpha_{2,m_2-1} n^{m_2-1}) r_2^n \\ & + \cdots + (\alpha_{t,0} + \alpha_{t,1} n + \cdots + \alpha_{t,m_t-1} n^{m_t-1}) r_t^n \end{aligned}$$

for $n = 0, 1, 2, \ldots$, where $\alpha_{i,j}$ are constants for $1 \leq i \leq t$ and $0 \leq j \leq m_i - 1$.

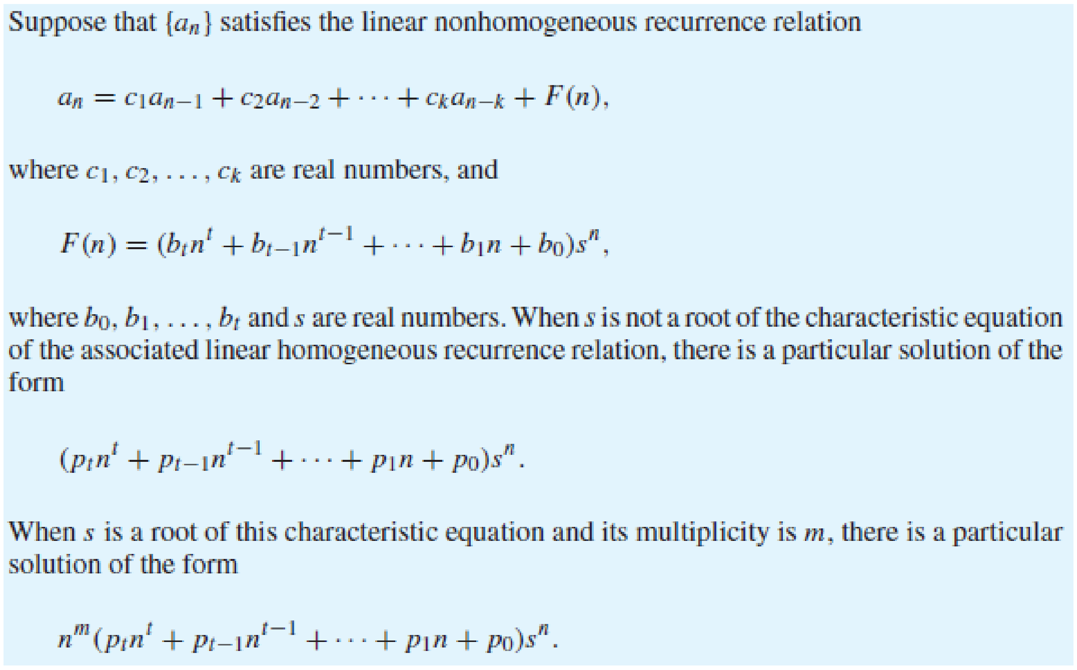
Again, we aren't going to worry about this general case, but I want you aware it exists.

Linear ***Non*homogeneous** Recurrence Relations with Constant Coefficients

Consider (the fairly common) case of a nonhomogeneous relation. That is, we add something that isn't a term of a_n but might be a function of n. So $a_n = c_1a_{n-1} + c_2a_{n-2} + \cdots + c_ka_{n-k} + F(n)$. In order to solve this, we are going to take three steps:

- Solve the *associated homogeneous recurrent relation* (that is the one that doesn't have that F(n) term!)
- Find a "particular" solution based on F(n).
- Add those two results.

We know how to do the first thing. And finding the particular solution in general can be annoying, but for most polynomials we can make progress as follows:



Suppose that $\{a_n\}$ satisfies the linear nonhomogeneous recurrence relation

$$a_n = c_1a_{n-1} + c_2a_{n-2} + \cdots + c_ka_{n-k} + F(n),$$

where $c_1, c_2, \ldots, c_k$ are real numbers, and

$$F(n) = (b_tn^t + b_{t-1}n^{t-1} + \cdots + b_1n + b_0)s^n,$$

where $b_0, b_1, \ldots, b_t$ and s are real numbers. When s is not a root of the characteristic equation of the associated linear homogeneous recurrence relation, there is a particular solution of the form

$$(p_tn^t + p_{t-1}n^{t-1} + \cdots + p_1n + p_0)s^n.$$

When s is a root of this characteristic equation and its multiplicity is m, there is a particular solution of the form

$$n^m(p_tn^t + p_{t-1}n^{t-1} + \cdots + p_1n + p_0)s^n.$$

In the general case, we'll have the situation that s=1. But that means we need to watch out if the homogeneous solution has a root that is 1 (which seems to happen a lot more than you'd expect).

Worked Example (Towers of Hanoi)

Let's compute $a_n=2a_{n-1}+1$ with $a_1=1$ (recall this is what we had for Towers of Hanoi).

The associated HRR is r-2=0, so r=2. So the homogeneous part is 2^n. Now we need the particular part.

When dealing with the particular solution $F(n)=1*1^n$, so "s" doesn't share a root with the characteristic equation ($1\neq2$). So our solution is of the form p_0 (a constant). Thus we've got $a_n=2^n+p_0$. Solving for the initial case we get $a_0=1=2^0+p_0$. So $p_0=-1$. And we've now found, rather than guessed, the closed form for the recurrence relation for Towers of Hanoi.

Divide and Conquer and the associated recurrence relations (8.3)

Let's look at a few fun algorithms that are what we call "divide and conquer"-type algorithms.

> We introduced a **binary search** algorithm in Section 3.1. This binary search algorithm reduces the search for an element in a search sequence of size n to the binary search for this element in a search sequence of size n/2, when n is even. (Hence, the problem of size n has been reduced to one problem of size n/2.) Two comparisons are needed to implement this reduction (one to determine which half of the list to use and the other to determine whether any terms of the list remain). (page 528)

If f(n) is the run time to find an element in a sorted list of n elements (assume n is a power of 2), how many comparisons do we need? Write this as a recurrence relation.

Consider a **merge sort** where we split the list in half, sort each half, and then merge the two sorted lists. What is the recurrence relation that describes the runtime of this algorithm? Assume it takes "n" steps to merge two lists each of size (n/2)[i]

[i] Why is "n" a reasonable number of steps to merge two *sorted* lists of size n/2 into a sorted list of "n" items?

The Master Theorem

As we often don't care about specific values, just the "order" of the complexity of the algorithm, we can generally get away without solving these recurrence relations—we are instead happy if we know the order of the solution. And it turns out, there is a "plug-and-chug" way to do this: The Master Theorem.

MASTER THEOREM Let f be an increasing function that satisfies the recurrence relation

$$f(n) = af(n/b) + cn^d$$

whenever $n = b^k$, where k is a positive integer, $a \geq 1$, b is an integer greater than 1, and c and d are real numbers with c positive and d nonnegative. Then

$$f(n) \text{ is } \begin{cases} O(n^d) & \text{if } a < b^d, \\ O(n^d \log n) & \text{if } a = b^d, \\ O(n^{\log_b a}) & \text{if } a > b^d. \end{cases}$$

Examples

1. Using the above, what are the values of a, b, c and d for the binary search algorithm? What is that algorithm's complexity?

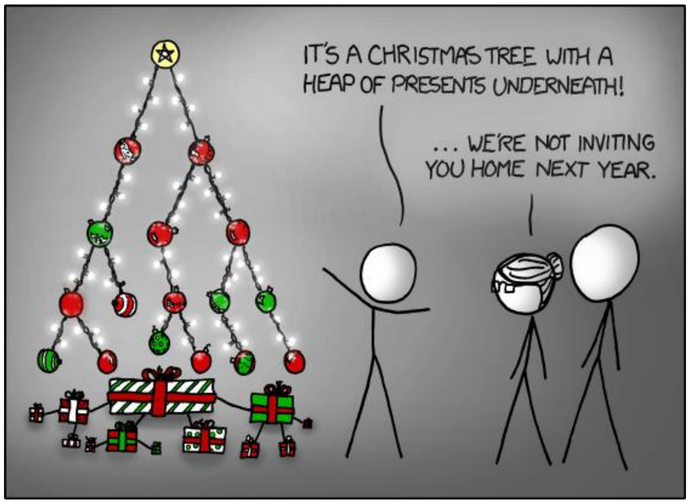
2. Using the above, what are the values of a, b, c and d for the merge sort algorithm? What is that algorithm's complexity?

3. Consider the relation F(N)=3F(N/3)+N (which is a merge sort split into 3 parts).

A graph **G=(V,E)** consists of:

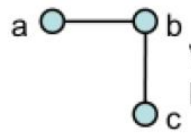
- a non-empty set V of vertices (or nodes)
- a set E of edges
    - Each edge is associated with two vertices (possibly equal).

- In a directed graph: all edges are directed.
- Undirected graph: all edges are undirected.
- Mixed graph: some of each.


IT'S A CHRISTMAS TREE WITH A HEAP OF PRESENTS UNDERNEATH!
... WE'RE NOT INVITING YOU HOME NEXT YEAR.

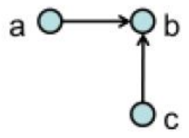
Some graphs:

- Undirected Graph



V = {a, b, c}
E = {{a,b}, {b,c}}

- Directed graph



V = {a, b, c}
E = {(a,b), (c,b)}

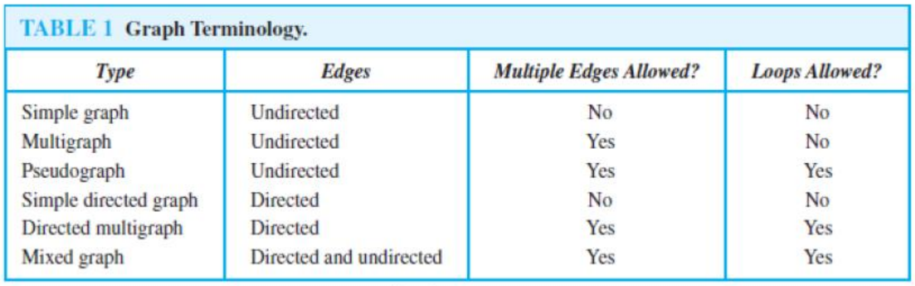
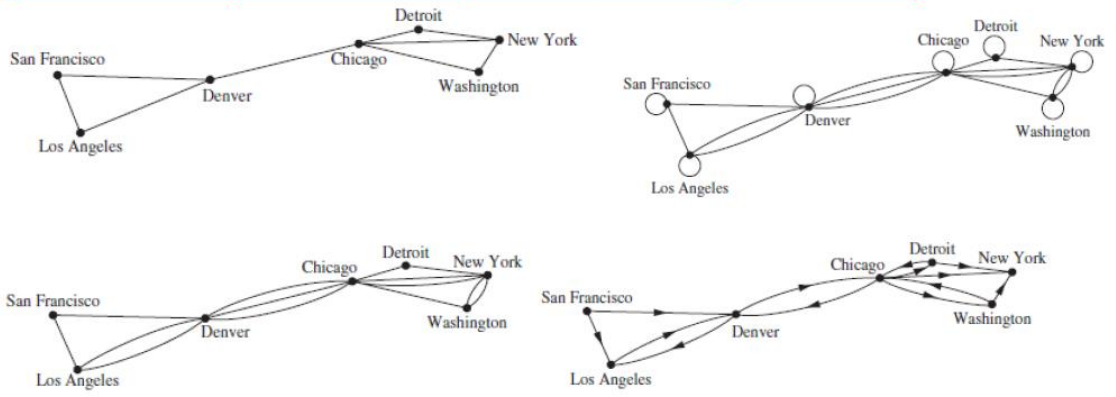
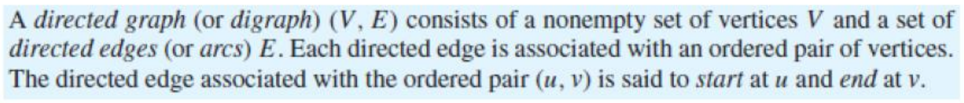


TABLE 1 Graph Terminology.

Type	*Edges*	*Multiple Edges Allowed?*	*Loops Allowed?*
Simple graph	Undirected	No	No
Multigraph	Undirected	Yes	No
Pseudograph	Undirected	Yes	Yes
Simple directed graph	Directed	No	No
Directed multigraph	Directed	Yes	Yes
Mixed graph	Directed and undirected	Yes	Yes

"loops allowed" means that an edge can connect a node to itself.



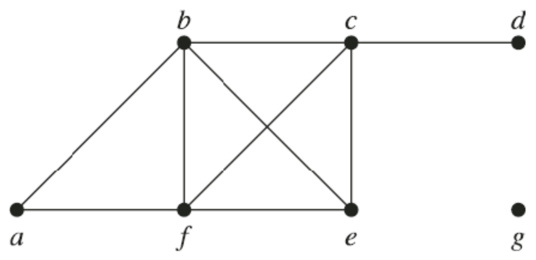


A *directed graph* (or *digraph*) (V, E) consists of a nonempty set of vertices V and a set of *directed edges* (or *arcs*) E. Each directed edge is associated with an ordered pair of vertices. The directed edge associated with the ordered pair (u, v) is said to *start* at u and *end* at v.

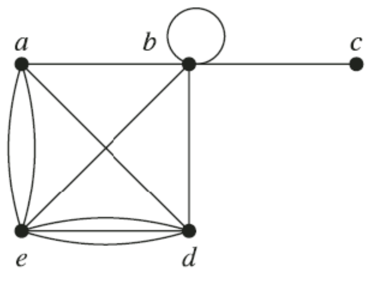
Basic Terminology

- Vertices u and v are **adjacent** if they are connected by an edge e
  - In an undirected graph: e = {u,v}
  - In a directed graph: e = (u,v)
- The **degree** of a vertex deg(v) is the number of "edge-ends" incident to it.
  - A self-loop edge is counted twice.
  - In a directed graph, distinguish:
    - **in-degree** deg−(v): number of incoming edges
    - **out-degree** deg+(v): number of outgoing edges
- A **subgraph** of a graph G = (V,E) is a graph H = (W,F) where W ⊆ V and F ⊆ E.
  - A subgraph H is a **proper subgraph** if H ≠ G.
- The **union of two simple graphs** G1 = (V1 , E1) and G2 = (V2, E2) is the simple graph with vertex set V1 ∪ V2 and edge set E1 ∪ E2.
  - The union of G1 and G2 is denoted by G1 ∪ G2.
- Two vertices u and v in an undirected graph G are called **adjacent** (or neighbors) in G if u and v are endpoints of an edge e of G. Such an edge e is called **incident** with the vertices u and v and e is said to **connect** u and v.
- The set of all neighbors of a vertex v of G = (V ,E), denoted by N(v), is called the **neighborhood** of V. If A is a subset of V , we denote by N(A) the set of all vertices in G that are adjacent to at least one vertex in A.

Examples and Questions:



G



H

1. What is the degree of each node?
2. Draw the graph A=(V,E) where V={a, b, c, d}, E={{a,b}, {b,d}}

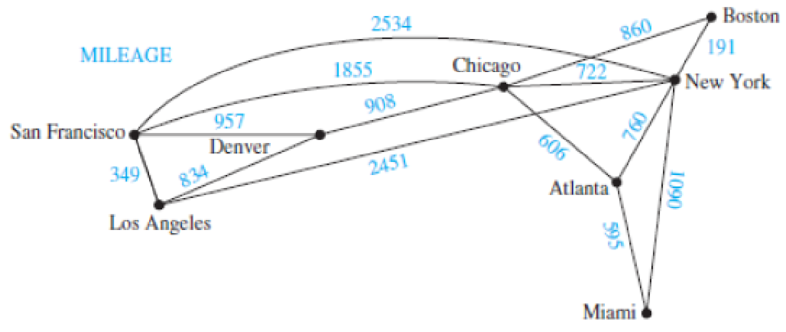
3. Draw the graph B=(V,E) where V={a, b, c, d}, E={{a,b}, {d,b}, {c,a}}

4. We now have 4 graphs total (A, B, G, H). Are any of them subgraphs of the other?
5. Draw the graph X=(V,E) where V={a, b, c, d}, E={(a,b), (b,d)}

Graph problems: paths and shortest paths

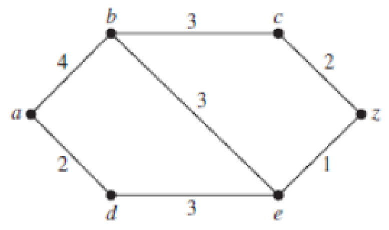
> Informally, a path is a sequence of edges that begins at a vertex of a graph and travels from vertex to vertex along edges of the graph. As the path travels along its edges, it visits the vertices along this path, that is, the endpoints of these edges. (p 678)

One interesting thing to do with paths is to find the shortest path between two vertices. Sometimes we put weights on the edges (so associate a number with each edge) and "charge" that amount for using the edge. We also want to be able to find the shortest path there.



A graph with weighted edges

What do you suppose is the order of complexity for finding shortest path?
We'll develop an algorithm for doing that next time.



Using graphs for something useful

Graphs are an extremely handy structure in practice.

Six Degrees of Kevin Bacon

This is a parlor game wherein movie buffs challenge each other to find the shortest path between an arbitrary actor and prolific Hollywood character actor Kevin Bacon.

Getting to Kevin Bacon's star on the Walk of Fame

Another thing you might want to do is get to Kevin Bacon's star on the Walk of Fame.

Figure 2: Map of the Walk of Fame in Hollywood

To people without some discrete mathematics background, the only two things these two problems would seem to have in common is, well, Kevin Bacon. But a solid CS person would also note that these are both graph theory problems. When solving the Six Degrees of Kevin Bacon or having Google Maps get you to Kevin Bacon's star, the problem is generally described as a graph and the goal is to find the shortest (weighted) path between two vertices in that graph.

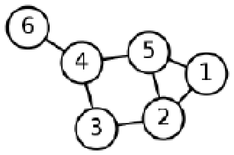


Figure 3: A graph (from pdx.edu)

What's really cool is that we can use the same algorithms to solve either of these two problems! And that's what this class should bring you. A worldview that lets you quickly see ways to address and solve problems you'll encounter as a CS or CE student.

You may recall that on the first day of class, I used graphs to illustrate how two problems that look to be quite different can be represented in the same way.

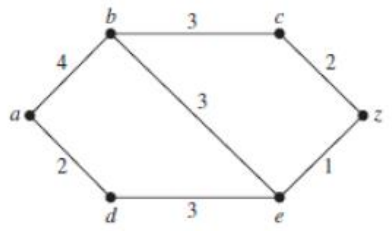
In this case, we are looking at two graphs. In both cases we are looking for the shortest path.

In the graph above, what is the shortest path from a to z?

How about from LA to Miami?

While it's pretty easy to find the shortest path from a to z by inspection, it's worth trying to formalize this into something we can make into an algorithm. Let's identify the "closest" note to a and then once we find it, use that to help find the second closest node etc.

Vertex	Distance
a	
d	



Let's be a bit more formal:

ALGORITHM 1 Dijkstra's Algorithm.

```
procedure Dijkstra(G: weighted connected simple graph, with
        all weights positive)
{G has vertices $a = v_0, v_1, \ldots, v_n = z$ and lengths $w(v_i, v_j)$
        where $w(v_i, v_j) = \infty$ if $\{v_i, v_j\}$ is not an edge in G}
for $i := 1$ to $n$
        $L(v_i) := \infty$
$L(a) := 0$
$S := \emptyset$
{the labels are now initialized so that the label of a is 0 and all
        other labels are $\infty$, and S is the empty set}
while $z \notin S$
        $u :=$ a vertex not in S with $L(u)$ minimal
        $S := S \cup \{u\}$
        for all vertices v not in S
                if $L(u) + w(u, v) < L(v)$ then $L(v) := L(u) + w(u, v)$
                {this adds a vertex to S with minimal label and updates the
                labels of vertices not in S}
return $L(z)$ {$L(z) =$ length of a shortest path from a to z}
```