University of Illinois at Urbana-Champaign
Dept. of Electrical and Computer Engineering

ECE 220: Computer Systems & Programming

Review: Good Design

Many Ways to Measure Goodness

What is the metric of goodness for software?

- number of instructions executed?
- amount of memory required?
- time required to finish?
- energy consumed?
- "correct" answers?

Yes.

All of them matter. Sorry.

Welcome to Engineering.

Two Goals Guide Our Choices in Software Design

1. **simpler (or feasible) approach**
   - avoid unnecessary complexity
   - use clear and obvious techniques when possible
   - a simple design that does work is better than a complex design that may work
2. **easy to understand and test**
   - as easy as possible to read (structure, indentation, comments!)
   - organize functionality to enable both separate and system-wide testing

It's That Time Again

Time to help me, I mean.

I need coffee.

But first, I need food.

I have a map.

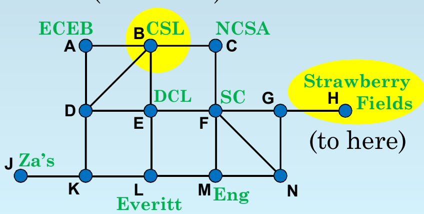
Help me to find my way
- from my office in CSL
- to Strawberry Fields.

Directions Needed from CSL to Strawberry Fields

I want to go from my office in CSL…

… to Strawberry Fields.

(from here)



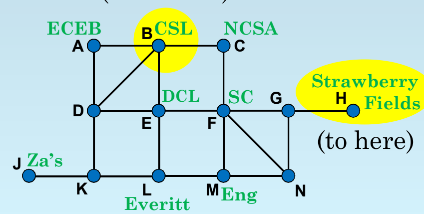
(to here)

Directions Needed from CSL to Strawberry Fields

Can you give me a path?

B → C or E → F → G → H? Great!

(from here)



(to here)

Actually, I May Need More Help

So.

Well.

Can you tag along with me?

Around campus, I mean?

We can give you a title.

"Assistant Walking Director"

Or something like that.

Actually… Just Teach My Computer!

Oh, wait!

You know how to program!

Teach my computer how to help me.

I will enter the map.

I will say where I am.

I will say where I want to go.

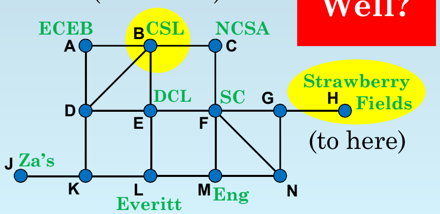
You tell it how to find a path.

Please Teach My Computer to Do What You Did

Here's the map. I want to be able to **find a path from any node to any other**.

(from here)

Well?



(to here)

Here's One Approach to Finding the Shortest Path

Don't panic!

I have **An Idea™**!

Here it is:

1. list all paths,
2. measure all of the paths, and
3. pick the shortest one.

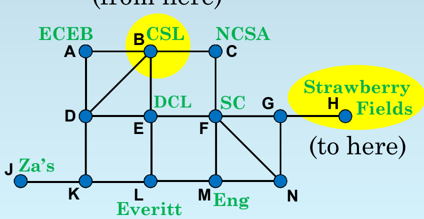
Sound good?

Let's Try My Idea: First, List the Paths

Starting from **B**…

B → A → B → A → B → A → B → A → B → A

(from here)



(to here)

Professors Shouldn't Be Allowed to Wander in Circles

Don't panic!

I have ~~**An Idea**~~™! **a better idea**

Here it is:

1. list ~~all paths~~, **all simple paths,**
2. measure all of the paths, and
3. pick the shortest one.

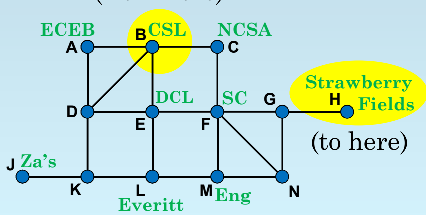
In mathematics, a "simple path" is one that includes any node at most one time. (So we can't go back to a place we've been already.)

Let's Try Again: First, List the Simple Paths

Starting from **B**… **That's one path!**

B → C → F → E → L → M→ N → G → H

(from here)



This Problem May Be a Little Exhausting

As it turns out …

For a general graph with **P** nodes,
- the number of simple paths
- between any pair of nodes
- is exponential in **P**.

The solution?

For **HW1**, please list **all paths for all pairs** of nodes in my map.

(Just kidding.)

One More Try! Let's Use a Queue

Let's make
- **a queue** of nodes
- and keep track of the **best previous location** for each node.

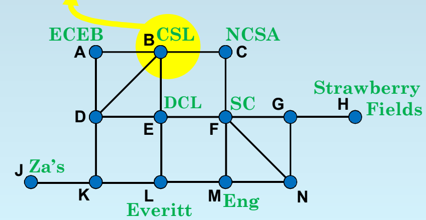
We'll **process the nodes** in the queue
- one by one
- **by adding any unvisited neighbors** to the queue.

Use a Queue to Find Shortest Paths

explored

queue	B										
previous	-										

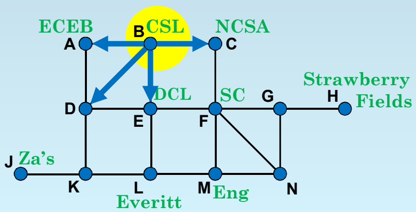
Add the starting node to the queue.



Process Node B by Adding all Neighbors

explored	√							
queue	B	A	D	E	C			
previous	-	B	B	B	B			

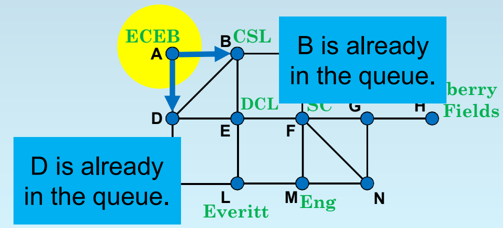
Process the next queue node by adding neighbors.



Process Node A by Adding all Neighbors

explored	√	√						
queue	B	A	D	E	C			
previous	-	B	B	B	B			

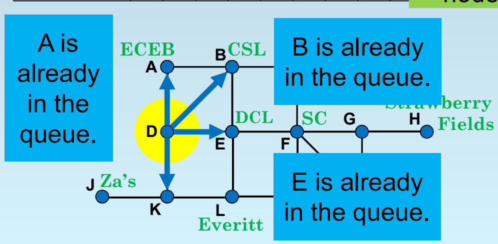
Process the next node.



Process Node D by Adding all Neighbors

explored	√	√	√					
queue	B	A	D	E	C	K		
previous	-	B	B	B	B	D		

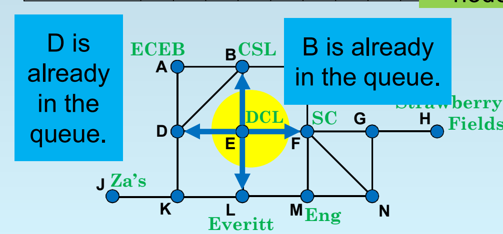
Process the next node.



Process Node E by Adding all Neighbors

explored	√	√	√	√				
queue	B	A	D	E	C	K	L	F
previous	-	B	B	B	B	D	E	E

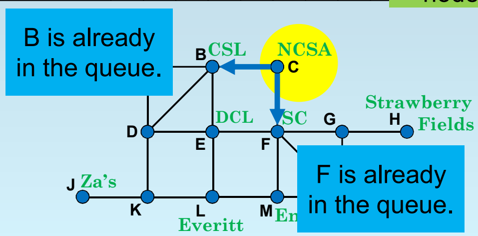
Process the next node.



Process Node C by Adding all Neighbors

explored	√	√	√	√	√					
queue	B	A	D	E	C	K	L	F		
previous	-	B	B	B	B	D	E	E		

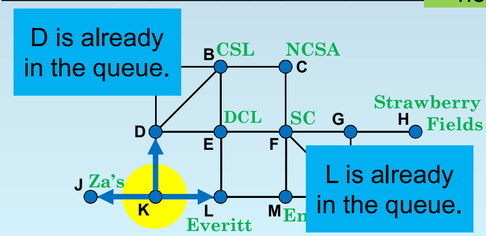
Process the next node.



Process Node K by Adding all Neighbors

explored	√	√	√	√	√	√				
queue	B	A	D	E	C	K	L	F	J	
previous	-	B	B	B	B	D	E	E	K	

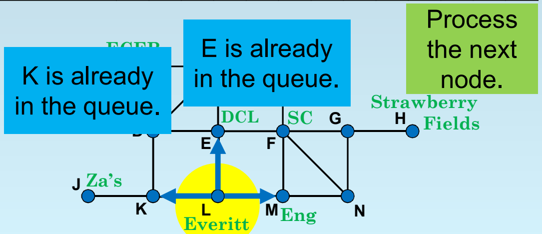
Process the next node.



Process Node L by Adding all Neighbors

explored	√	√	√	√	√	√	√						
queue	B	A	D	E	C	K	L	F	J	M			
previous	-	B	B	B	B	D	E	E	K	L			

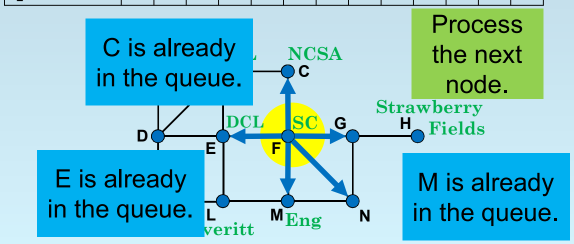
Process the next node.



Process Node F by Adding all Neighbors

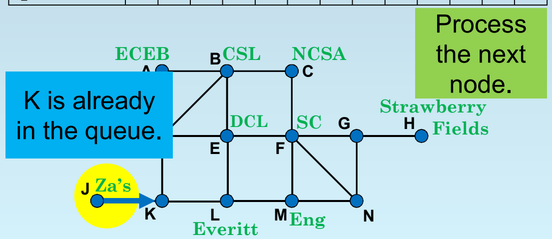
explored	√	√	√	√	√	√	√	√					
queue	B	A	D	E	C	K	L	F	J	M	N	G	
previous	-	B	B	B	B	D	E	E	K	L	F	F	

Process the next node.



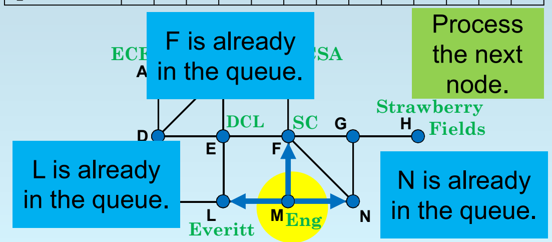
Process Node J by Adding all Neighbors

explored	√	√	√	√	√	√	√	√	√				
queue	B	A	D	E	C	K	L	F	J	M	N	G	
previous	-	B	B	B	B	D	E	E	K	L	F	F	



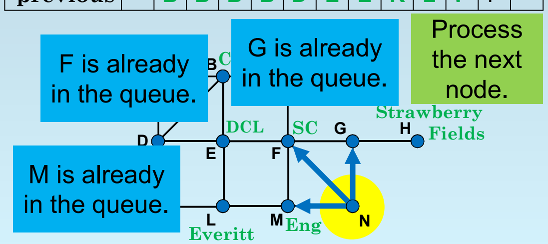
Process Node M by Adding all Neighbors

explored	√	√	√	√	√	√	√	√	√	√			
queue	B	A	D	E	C	K	L	F	J	M	N	G	
previous	-	B	B	B	B	D	E	E	K	L	F	F	



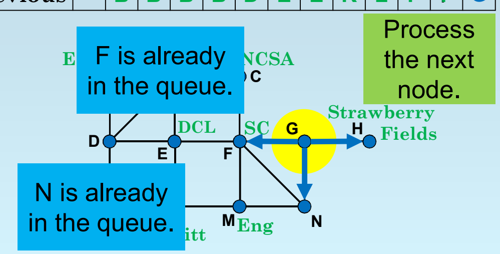
Process Node N by Adding all Neighbors

explored	√	√	√	√	√	√	√	√	√	√	√		
queue	B	A	D	E	C	K	L	F	J	M	N	G	
previous	-	B	B	B	B	D	E	E	K	L	F	F	



Process Node G by Adding all Neighbors

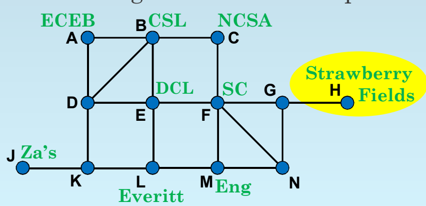
explored	√	√	√	√	√	√	√	√	√	√	√	√	
queue	B	A	D	E	C	K	L	F	J	M	N	G	H
previous	-	B	B	B	B	D	E	E	K	L	F	F	G



We Found Strawberry Fields!

queue	B	A	D	E	C	K	L	F	J	M	N	G	H
previous	-	B	B	B	B	D	E	E	K	L	F	F	G

We reached our goal! Now for the path…



Use the Previous Nodes to Find the Path Backwards

queue	B	A	D	E	C	K	L	F	J	M	N	G	H
previous	-	B	B	B	B	D	E	E	K	L	F	F	G

How did we get to H? From G.

How did we get to G? From F.

How did we get to F? From E.

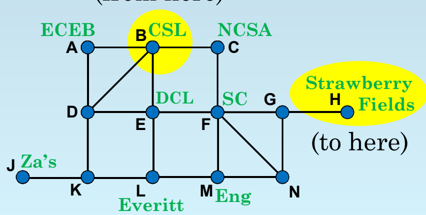
How did we get to E? From B.

Now We Have a Way to Find Shortest Paths!

So we found …

B → E → F → G → H? Great!

(from here)



(to here)

Breadth-First Search Finds Short Paths Quickly

distance	0	1				2			3				4
queue	B	A	D	E	C	K	L	F	J	M	N	G	H
previous	-	B	B	B	B	D	E	E	K	L	F	F	G

The approach that we used is called **breadth-first search (BFS)**.

It explores nodes **in order of distance** (see the line on top of our queue).

So you can use BFS with a commercial map database and still find a path just as quickly.

Is BFS Artificial Intelligence?

BFS was invented by E. F. Moore (remember Moore machines?).

After you have seen BFS, it seems pretty simple.

But people used to think of it as "artificial intelligence."

You will use DFS (depth-first, using a stack instead of a queue) soon in an MP.

The Point: Algorithms Can Be Subtle

Finding the simplest algorithm to solve a problem can be challenging.

Experience helps.

Classes help.

All CompEs must take **CS225 (Data Structures)** and **CS374 (Algorithms)**, so you have time to learn.

Don't worry about finding ideal solutions in our class.

Build Good Habits Now

What can you do now?

1. **Start with a mental model** (for now, as a flow chart on paper).
2. **Write lots of comments.**
3. **Structure your code clearly.**
4. **Avoid repetition:**
   - Reuse code instead of cutting and pasting.
   - Every time you copy code, you copy any bugs that the code contains.

Some Tips as You Get Started

5. **Design your code to simplify testing:**
   - You have finite time.
   - A program that is hard to test will be tested less often and less thoroughly.
6. **Test code piece by piece:**
   - Do not write all code before testing.
   - Bugs are more likely in new code.

