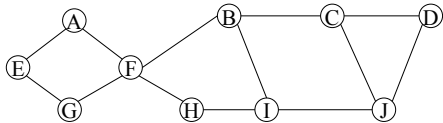


### Articulation Vertices

Suppose you are a hacker seeking to disrupt a company's intranet. Which router in the diagram below should you choose to blow up in order to cause the maximum amount of damage?



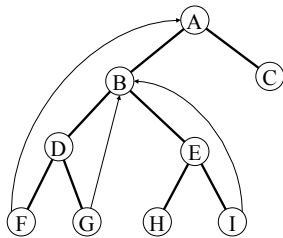
### Connectivity

The connectivity of a graph is the smallest number of vertices whose deletion will disconnect a graph. For graphs with an articulation vertex, the connectivity is one.

How can we test for 0 or 1 connectivity by brute force?

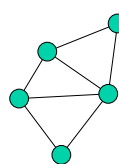
Is there a better algorithm we can use?

### Finding Articulation Vertices with DFS



### Minimum Height Spanning Trees

Both DFS and BFS produce spanning trees, but how can we guarantee that we find one of minimal height?



What height will BFS give if we start from the top node?

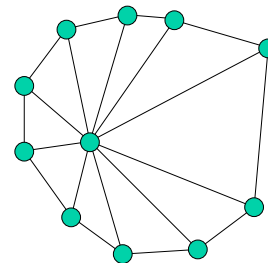
Can we do better than this?

### Weighted Graphs

A weighted graph is one in which each edge has a "cost" of some kind associated with it. Many algorithms become more complex when in addition to finding *any* solution, we also want to minimize the cost of that solution.

Example: it is easy to find a spanning tree (using BFS or DFS) and even the minimum height spanning tree, but how hard is it to find the minimum *cost* spanning tree? (typically this is simply called the "minimum spanning tree")

### Example Graph



## Prim's Algorithm

```

Prim-MinSpanningTree(G) {
  Select an arbitrary vertex to start
  the tree with;
  While(non-tree vertices exist){
    Select the edge of minimum weight
    between a tree and non-tree vertex;
    Add the selected edge to the tree
    and mark the new vertex as being in
    the tree ;
  }
}

```

## Kruskal's Algorithm

```

Kruskal-MinSpanningTree(G) {
  Put the edges in a priority queue
  ordered by weight;
  count = 0;
  while (count < n-1) {
    get next edge (v,w);
    if (component (v) != component(w));
    merge (component(v) , component(w));
    count++;
  }
}

```

## Shortest Paths

The shortest path between two vertices  $s$  and  $t$  in an unweighted graph can be constructed using a breadth-first search from  $s$ . When we first encounter  $t$  in the search, we will have reached it from  $s$  using the minimum number of possible edges.

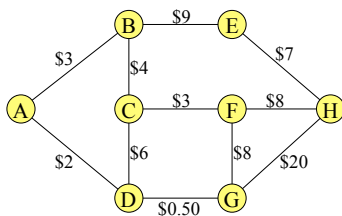
This is only true if all edges have the same weight; when they have differing weights we need to do something slightly more complex.

## Dijkstra's Algorithm

Dijkstra's algorithm is another variation on Breadth-First-Search where, like minimum spanning trees, a priority queue determines which vertex is to be explored next. In this case, the priority of each vertex is the minimum total path found thus-far to that vertex.

The principle behind Dijkstra's algorithm is that given the shortest path between vertices  $x$  and  $y$ , any vertex  $v_i$  on that path must bisect it into the shortest path from  $x$  to  $v_i$ , and the shortest path from  $v_i$  to  $y$ .

## Dijkstra Example



## All-Pairs Shortest Path

What if we want to know the shortest path between all pairs of vertices in a weighted graph? How long will this take?

### Example Problem

At an arm wrestling competition, one of the organizers is told to setup the order of contestants entering the awards ceremony such that no one who lost a competition comes into the room before the person who beat them.

Assuming that in every case the stronger person won, and no two people are the exact same strength, *but* not every pair of people arm-wrestled, how can he come up with a good ordering for them to enter in?

### Example Problem

Marleen Thompson was taking the sixth grade class that she teaches to the zoo on a field trip. As part of the school policy, the students need to be broken up into small groups (of at least two) that must stay together at all times.

Each of the students gives her a list of other students that they are willing to pair with. How can she group the students into as many groups as possible, where each is with at least one other student on their list? Can we do this using graph theory?

### Example Problem

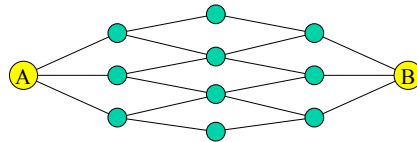
One way that professors found to detect students cheating on programming assignments is to study the distances between each pair of programs, and picking out those that had very similar code for further examination.

To get around this, groups of cheaters banded together and a couple of people wrote each part, and then the sections of the programs were mixed-and-matched, so that while two programs might look similar, they were not close enough to be sure of cheating.

What should the professors do now?

### Example Problem

A company has all of its branches in a particular city all wired directly to each other to maximize the rate that data can be transferred around. If a critical operation needs to transfer information from A to B, what is the fastest it can be done given each connection has its own rate?

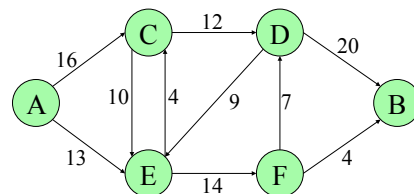


### Network Flow

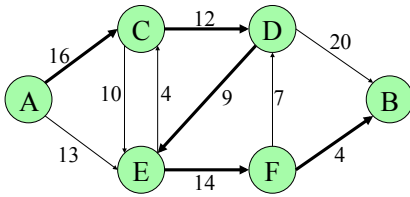
**Inputs:** A graph  $G$ , where each edge  $e = (i, j)$  has a capacity  $c_e$ . A source node  $A$ , and a sink node  $B$ .

**Problem:** What is the maximum flow you can route from  $A$  to  $B$  while respecting the capacity constraint of each edge?

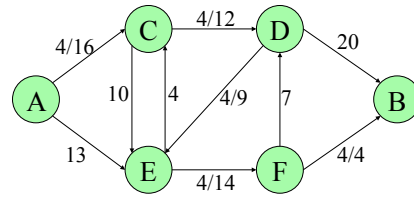
### Example



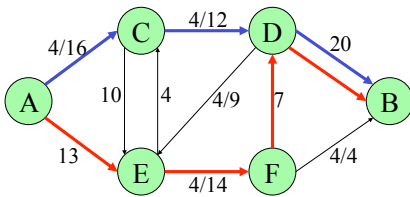
One Possible Path



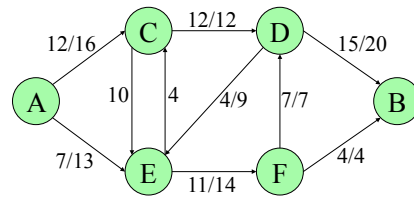
The capacity used so far...



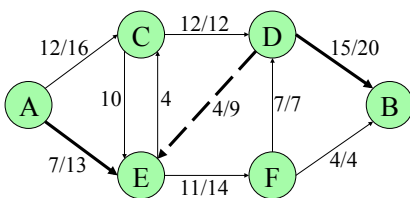
Two more paths...



Final Capacities?



One final possible path...



And the resulting flow...

