CSC 225 - Summer 2019

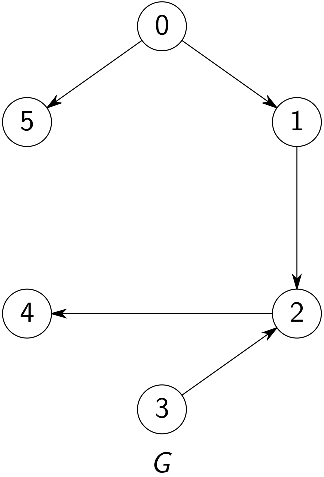
Transitive Closure

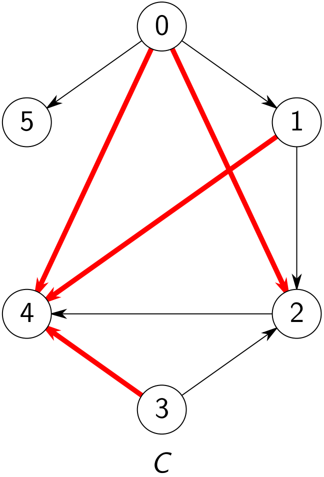
Bill Bird

Department of Computer Science
University of Victoria

July 30, 2019

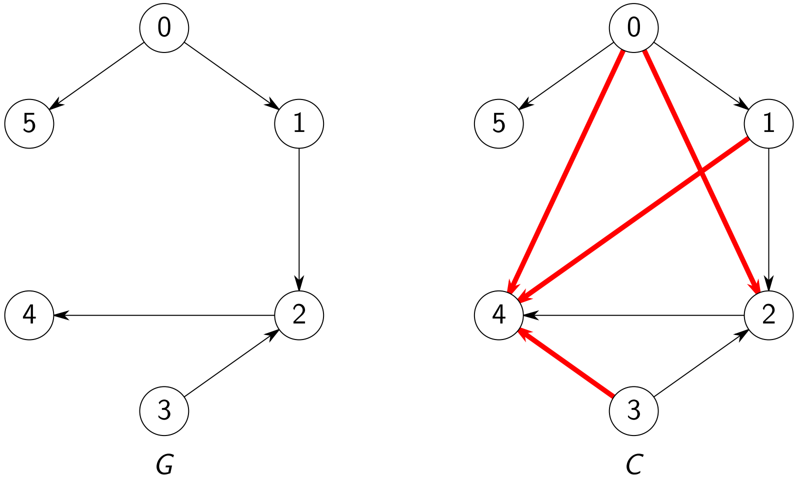
Transitive Closure (1)





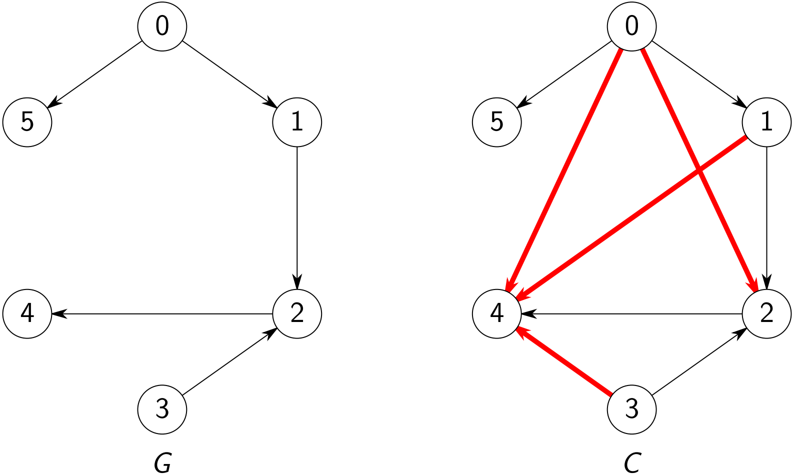
The **transitive closure** of a graph G is a graph C containing the vertices of G and an edge uv if v is reachable from u in G.

Transitive Closure (2)



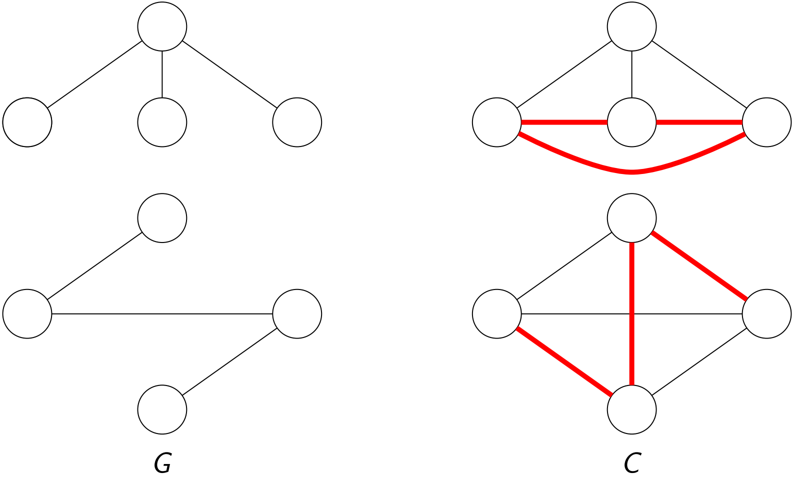
Transitive closure graphs are useful for efficiently representing the set of vertices reachable from every vertex.

Transitive Closure (3)



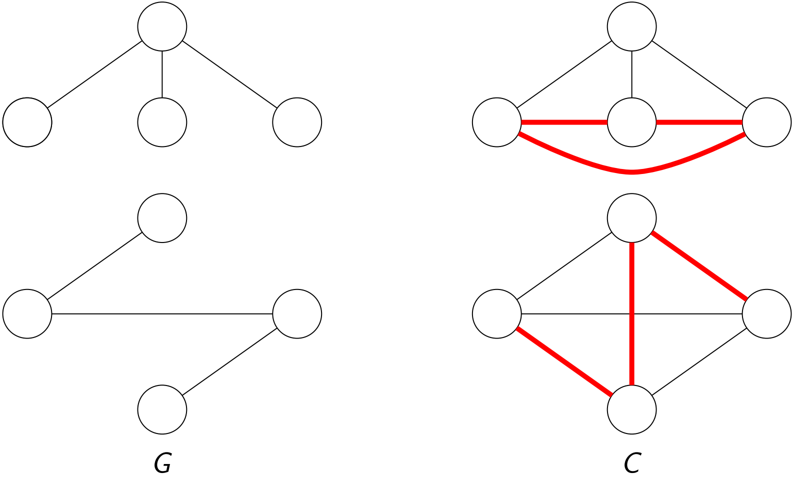
When stored in an adjacency matrix, a transitive closure can be used to look up reachability information in $\Theta(1)$ time.

Transitive Closure (4)



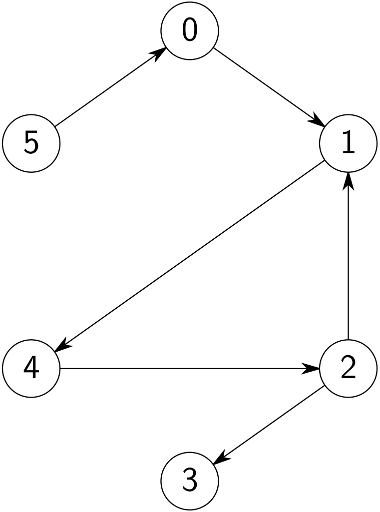
Transitive closures of undirected graphs convert each component into a complete graph.

Transitive Closure (5)



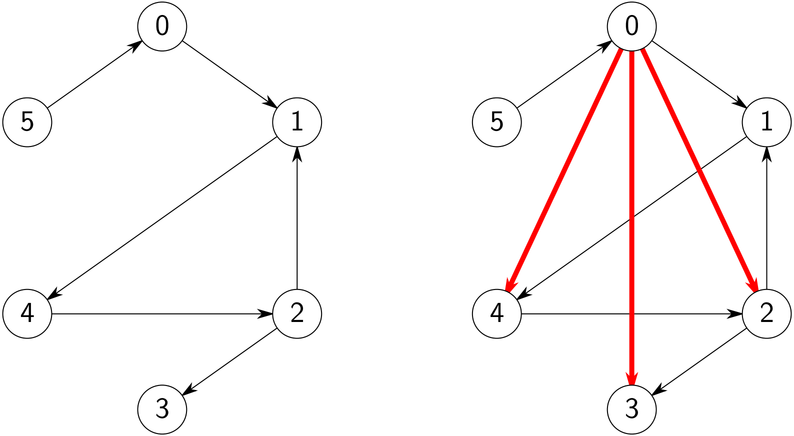
The transitive closure of an undirected graph can be computed in $\Theta(n^2)$ time by traversing each component and adding all possible edges between the vertices in the component.

Computing Transitive Closures (1)



Exercise: Compute the transitive closure of the graph above.

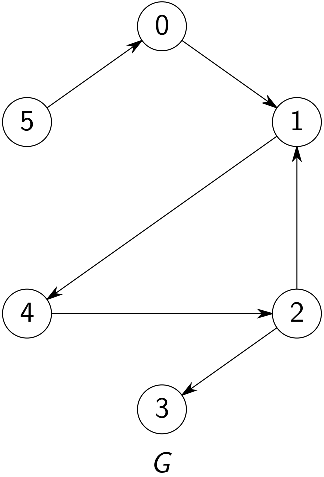
Computing Transitive Closures (2)

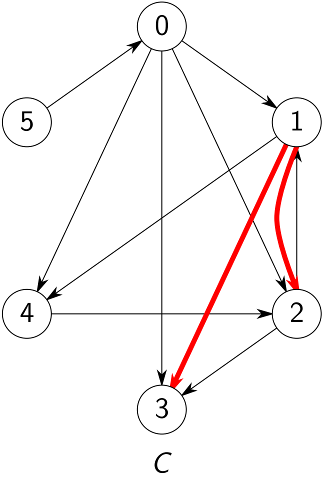


G C

One option is to iterate over the vertices of the graph and add edges from each vertex v to all the vertices reachable from v.

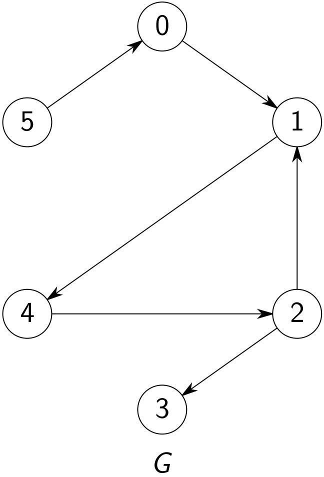
Computing Transitive Closures (3)

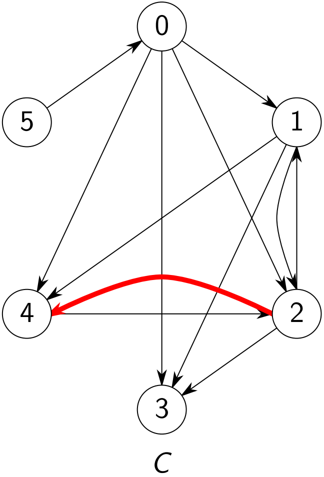




One option is to iterate over the vertices of the graph and add edges from each vertex v to all the vertices reachable from v.

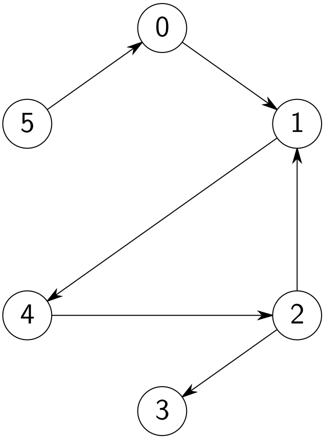
Computing Transitive Closures (4)



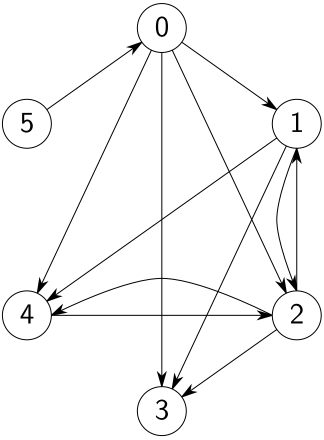


One option is to iterate over the vertices of the graph and add edges from each vertex v to all the vertices reachable from v.

Computing Transitive Closures (5)



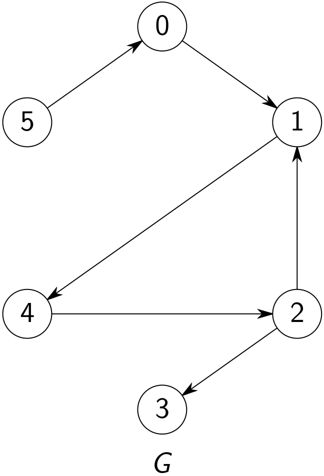
G

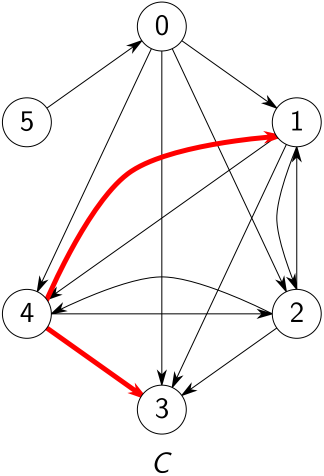


C

One option is to iterate over the vertices of the graph and add edges from each vertex v to all the vertices reachable from v.

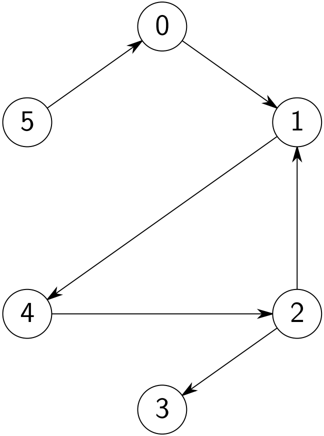
Computing Transitive Closures (6)



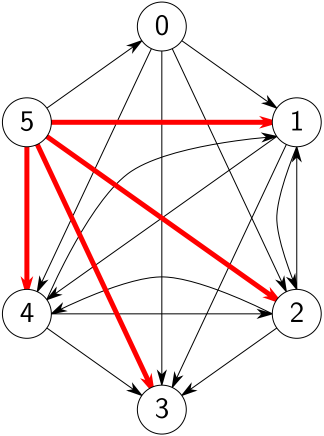


The set of vertices reachable from each vertex v can be found with a traversal rooted at v.

Computing Transitive Closures (7)



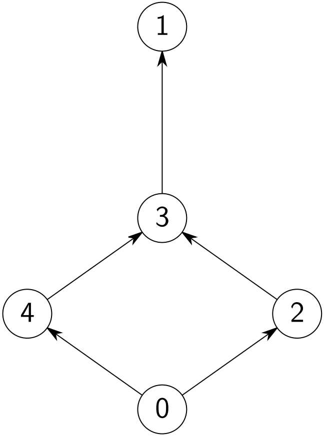
G



C

Computing the transitive closure of a directed graph with n traversals requires $\Theta(n(n+m))$ time.

Adjacency Matrices (1)

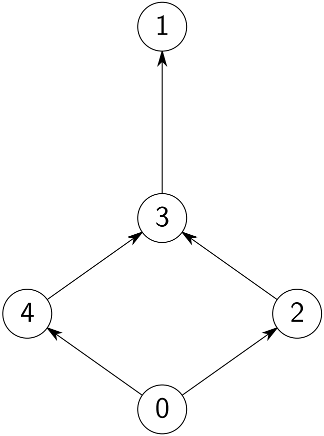


$$A$$

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

Note: The topic covered from this slide onward (linear algebra on adjacency matrices) will not be covered on the last exam (but will be relevant to CSC 226).

Adjacency Matrices (2)

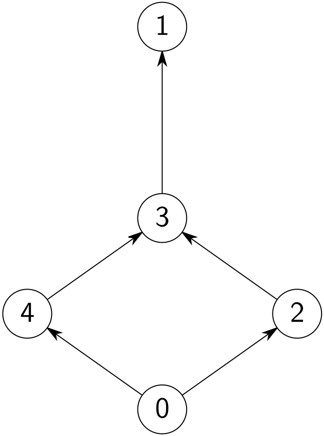


$$A$$

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

Consider the adjacency matrix of the graph above.

Adjacency Matrices (3)

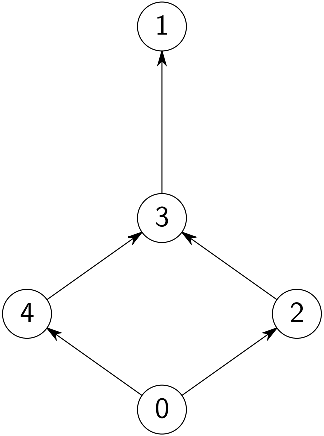


$$A$$

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

Entry $A_{i,j}$ of an adjacency matrix is 1 if there is an edge from vertex i to vertex j.

Adjacency Matrices (4)

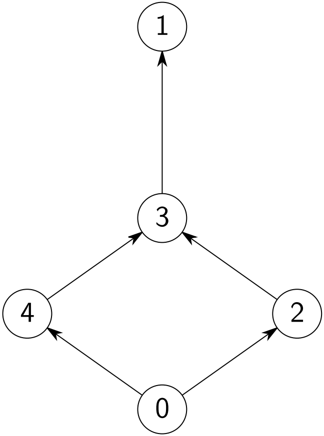


$$A$$

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

We can also view the value $A_{i,j}$ as counting the number of paths of length 1 from vertex i to vertex j.

Adjacency Matrices (5)

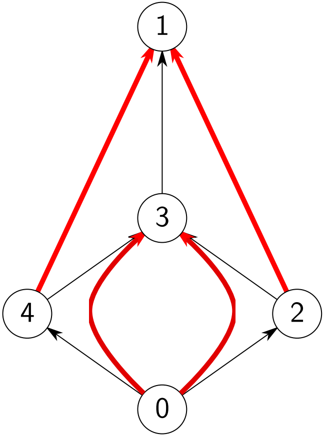


$$A$$

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

Strange Compulsion: Treat the adjacency matrix A like any other $n \times n$ matrix and try some linear algebra.

Adjacency Matrices (6)

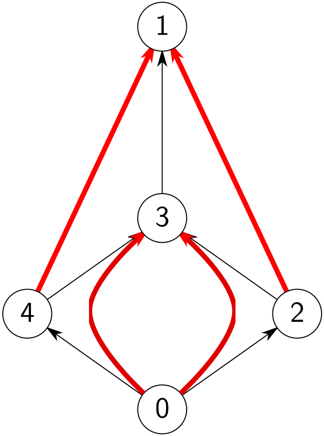


$$A^2$$

$$\begin{bmatrix} 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

The entries of the matrix product $A \cdot A = A^2$ correspond to walks of length 2 in the original graph.

Adjacency Matrices (7)

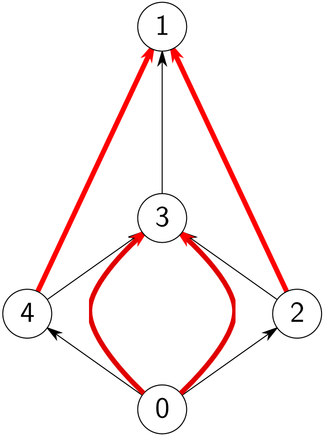


A^2

$$\begin{bmatrix} 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

(A **walk** is similar to a path, but is allowed to repeat vertices and edges)

Adjacency Matrices (8)

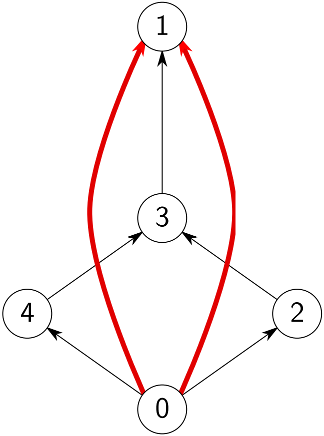


$$A^2$$

$$\begin{bmatrix} 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

The value $A^2_{0,3}$ is 2, since there are two directed walks of length 2 from vertex 0 to vertex 3.

Adjacency Matrices (9)

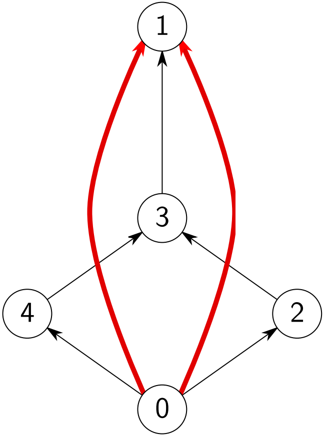


$$A^3$$

$$\begin{bmatrix} 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

In general, the entries of the matrix A^k will count walks of length exactly k.

Adjacency Matrices (10)

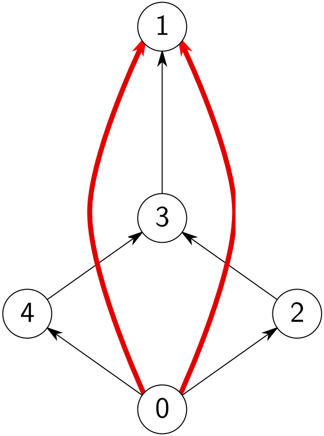


$$A^3$$

$$\begin{bmatrix} 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The value $A^3_{0,1}$ is 2 because there are 2 walks of length 3 from vertex 0 to vertex 1.

Adjacency Matrices (11)

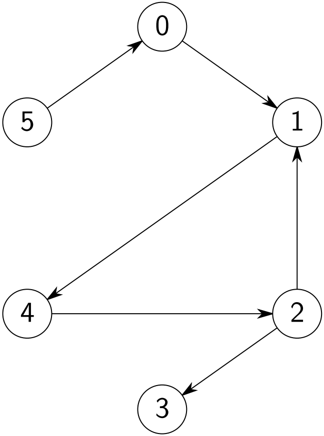


A^3

$$\begin{bmatrix} 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Observation: If $A^k_{i,j} > 0$ for any value of k, then vertex j is reachable from vertex i.

Transitive Closure With Matrices (1)

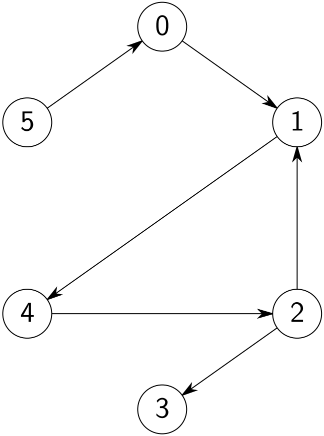


$$A$$

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

Matrix multiplication can be used to find the transitive closure of a graph.

Transitive Closure With Matrices (2)

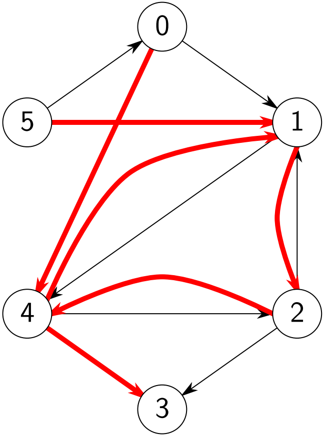


$$A$$

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

Start with the initial graph.

Transitive Closure With Matrices (3)

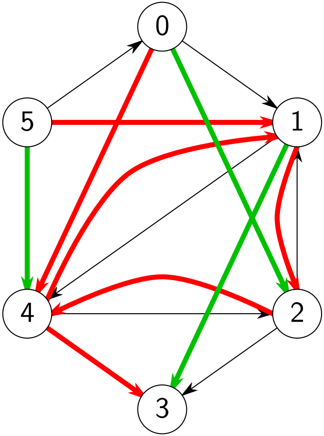


$$A^2$$

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

If A^2 has any non-zero entries, add the corresponding edges to G if they do not already exist.

Transitive Closure With Matrices (4)

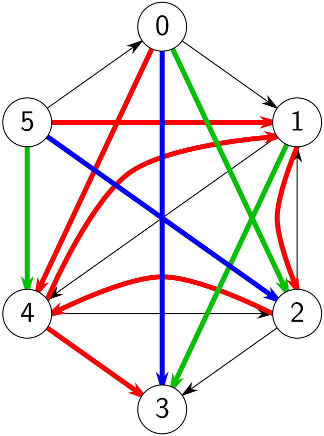


A^3

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

Similarly, add all edges corresponding to non-zero entries of A^3.

Transitive Closure With Matrices (5)

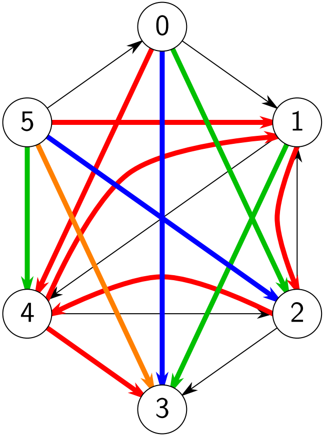


$$A^4$$

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

Continue computing matrix powers and adding edges until reaching A^{n-1}

Transitive Closure With Matrices (6)



$$A^5$$

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

Since the graph has 6 vertices, all paths must have length at most 5.

Transitive Closure With Matrices (7)

Let A be the adjacency matrix for a graph G. Let

$$M = A + A^2 + A^3 + \ldots + A^n$$

The transitive closure C of G will have an edge ij if and only if $M_{ij} > 0$. Multiplying two $n \times n$ matrices requires $\Theta(n^3)$ time in practice[1]. Computing the sum above for M therefore requires $\Theta(n^5)$ time.

[1] Extremely impractical algorithms for matrix multiplication can achieve $\Theta(n^{2.37})$ time.

Transitive Closure With Matrices (8)

The sum

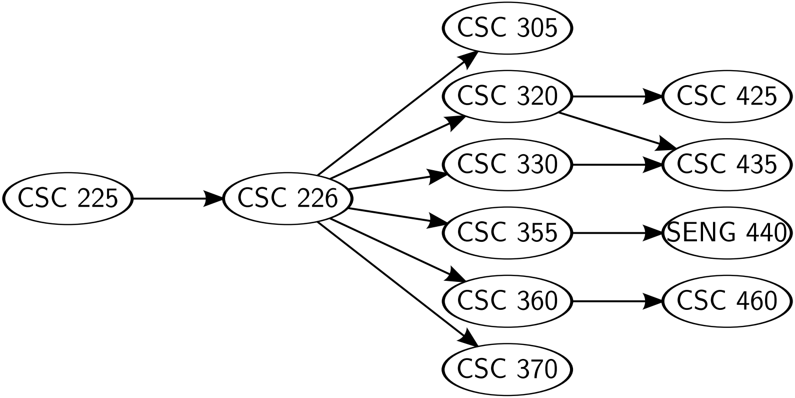
$$M = A + A^2 + A^3 + \ldots + A^n$$

can be rewritten (using Horner's rule) as

$$M = \underbrace{A(I + A(I + A(I + \ldots)))}_{n \text{ additions}}$$

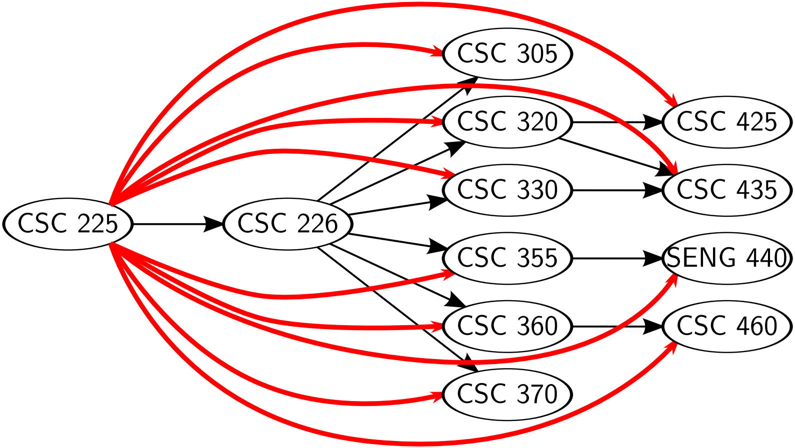
which requires n additions and n multiplications. This method still requires $\Theta(n^4)$ time.

One More Graph (1)

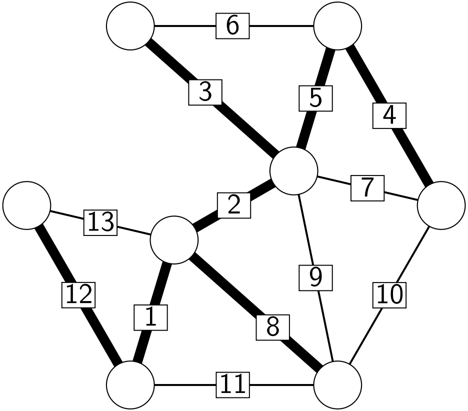


Consider the directed graph above.

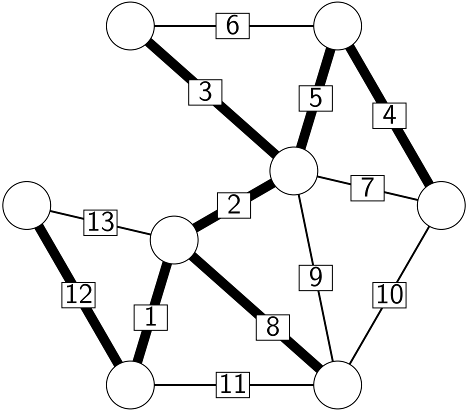
One More Graph (2)



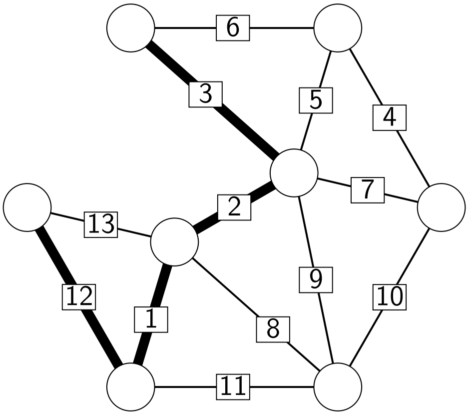
Question: Now that CSC 225 is over, what can you do with all of your newfound free time?



CSC 226: Algorithms and Data Structures II

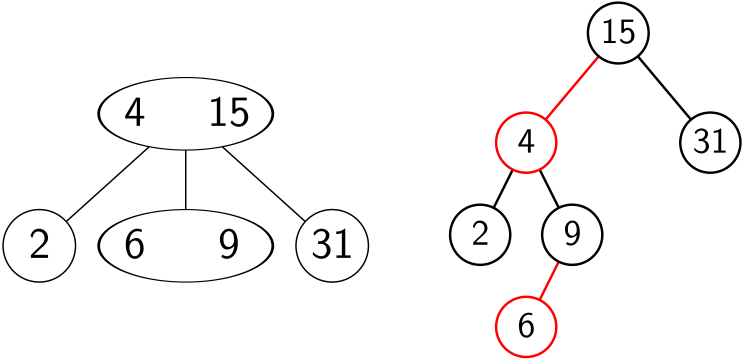


Algorithms for minimum weight spanning trees.



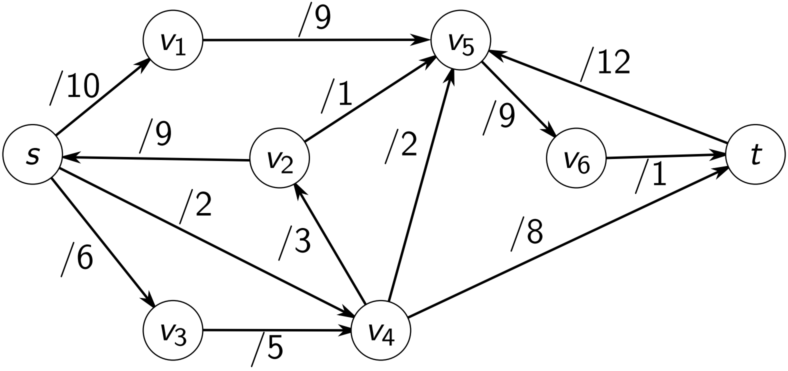
Algorithms for finding minimum weight paths between two vertices in an edge-weighted graph.

CSC 226 (4)

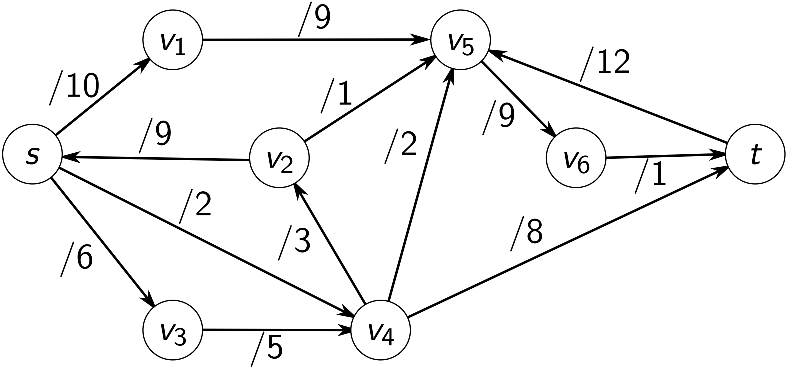


Balanced search trees, including AVL trees, 2-3 trees (left) and red-black trees (right).

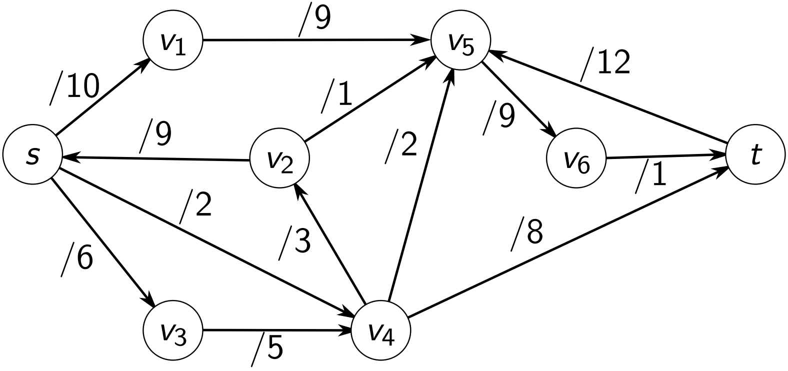
CSC 226 (5)



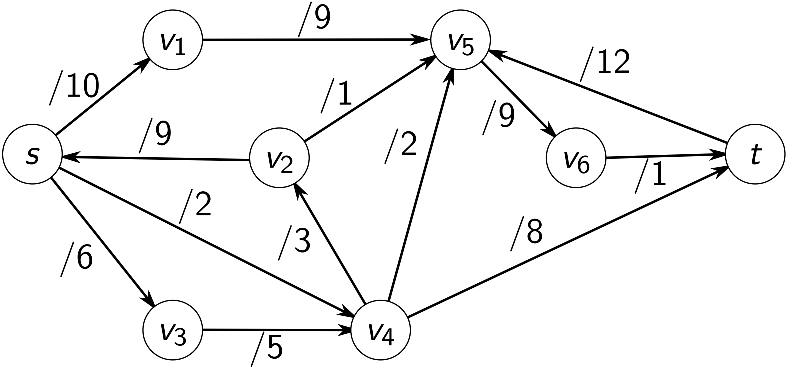
The graph above is a **flow network**.



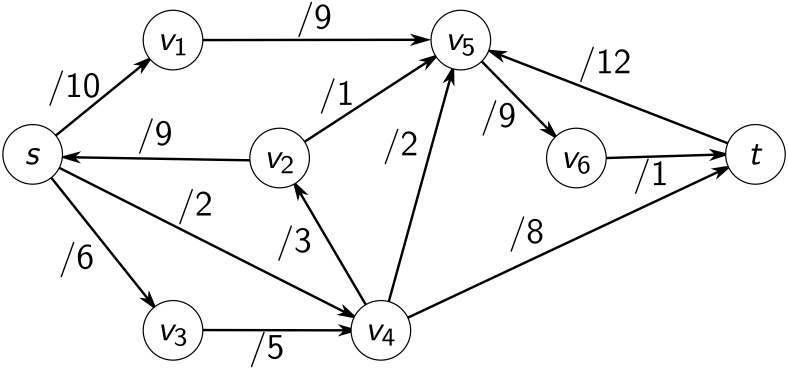
Consider a network of water pipes, where water is allowed to flow in one direction only.



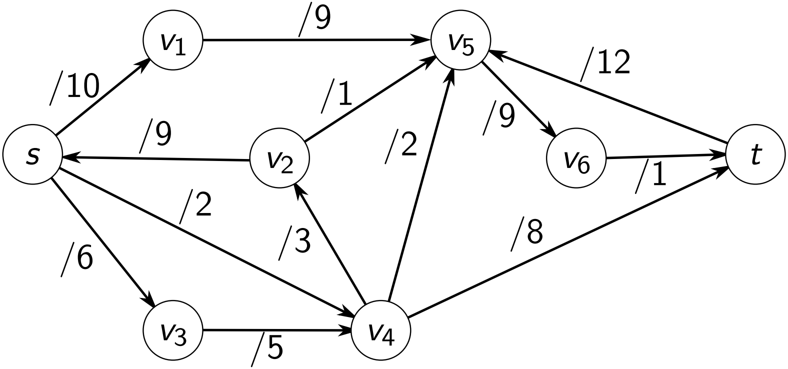
Each of the pipes has a maximum capacity (represented by edge weights).



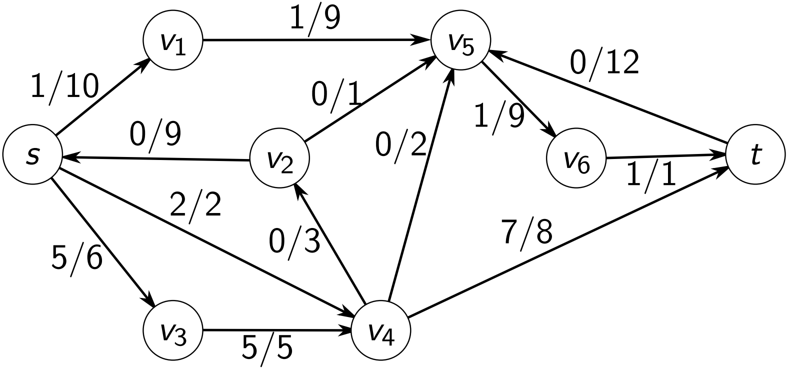
The vertex s is a **source** and the vertex t is a **sink**



Except for s and t, the amount of water entering each vertex must equal the amount leaving it (i.e. water flow is conserved).

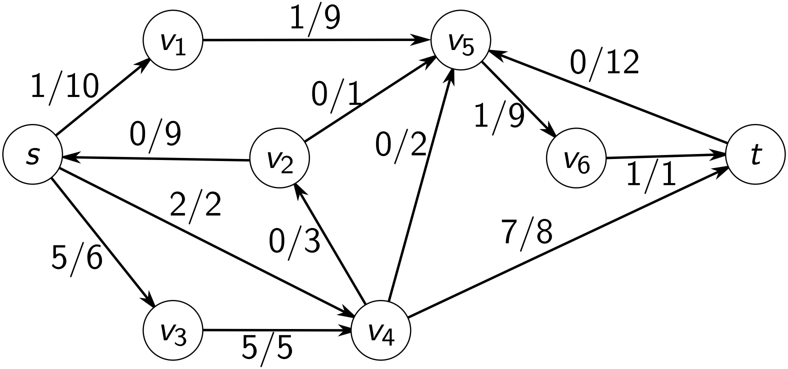


The **max flow** problem is to find an assignment of values to each edge which maximizes the total volume of water flowing from s to t.



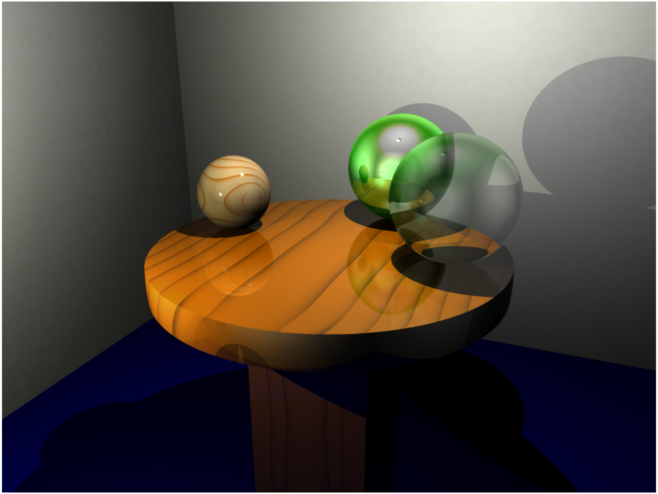
The assignment above is a maximum flow from s to t.
The total volume, or **value**, of the flow is 8.

CSC 226 (12)



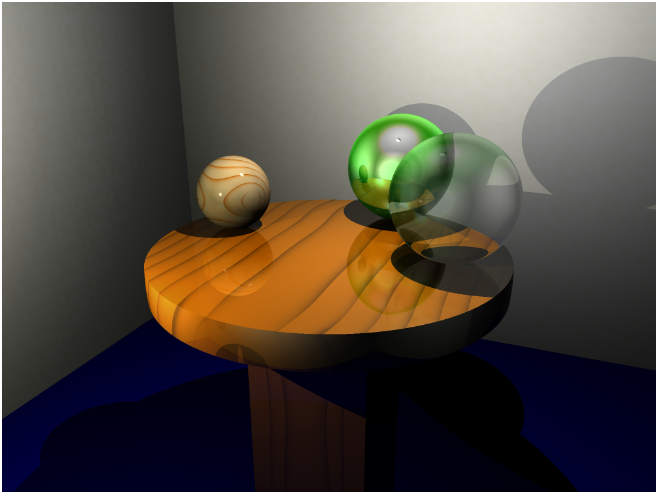
CSC 226 covers efficient algorithms to compute maximum flows in networks.

CSC 305 (1)



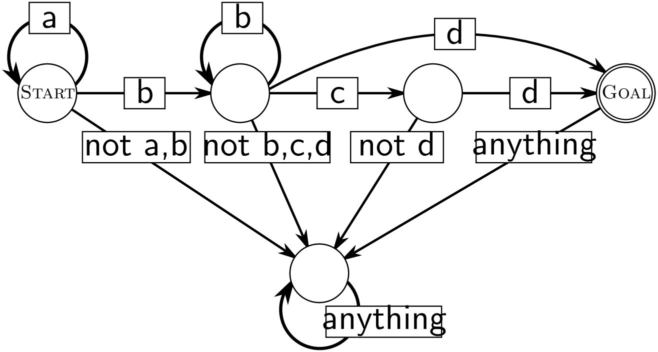
CSC 305: "Introduction" to Computer Graphics
Computational mathematics, hardware rendering, 3d graphics, ray tracing.

CSC 305 (2)



CSC 305: "Introduction" to Computer Graphics

3d graphics courses always involve writing a ray tracer (to generate images like the one above).



CSC 320: Foundations of Computer Science

Formal languages, finite automata (such as the DFA pictured above), computability and complexity.

```
let rec sumlist L =
    match L with
    | [] -> 0
    | [x] -> x
    | x :: tail ->
        x + sumlist tail

let A = [1; 2; 3; 4; 5; 6]
printfn "Sum(A) = %d" (sumlist A)
```

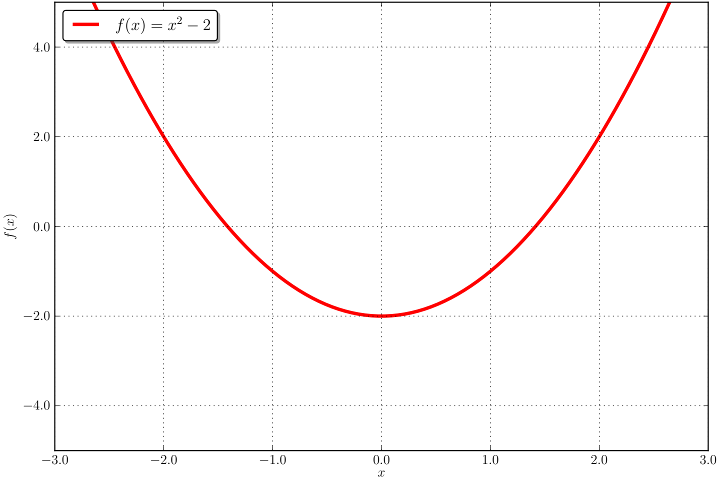
CSC 330: Programming Languages
Programming language design and semantics. Case studies of different programming languages.

CSC 330 (2)

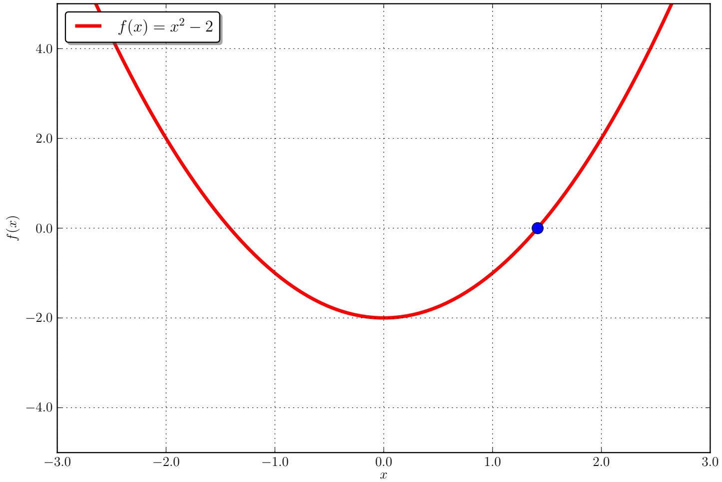
```
let rec sumlist L =
    match L with
    | [] -> 0
    | [x] -> x
    | x :: tail ->
        x + sumlist tail

let A = [1; 2; 3; 4; 5; 6]
printfn "Sum(A) = %d" (sumlist A)
```

CSC 330: Programming Languages
CSC 330 covers **functional programming languages**, such as F#.
The F# code above computes the sum of a list.

CSC 349a (1)

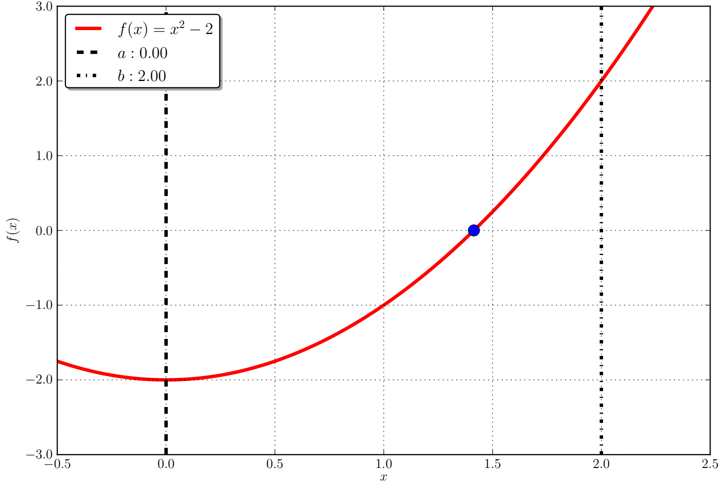


Problem: Compute $\sqrt{2}$ to 15 significant digits using only elementary arithmetic ($+$, $-$, $*$ and $/$).

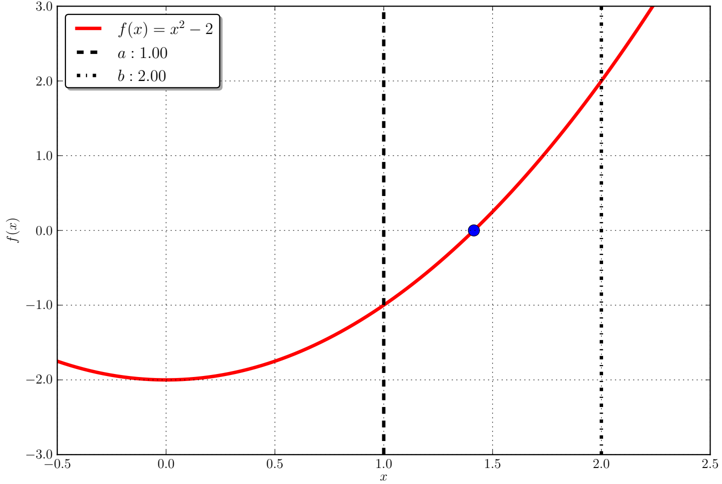


Observation: The value $x = \sqrt{2}$ is a root of the polynomial $f(x) = x^2 - 2$.

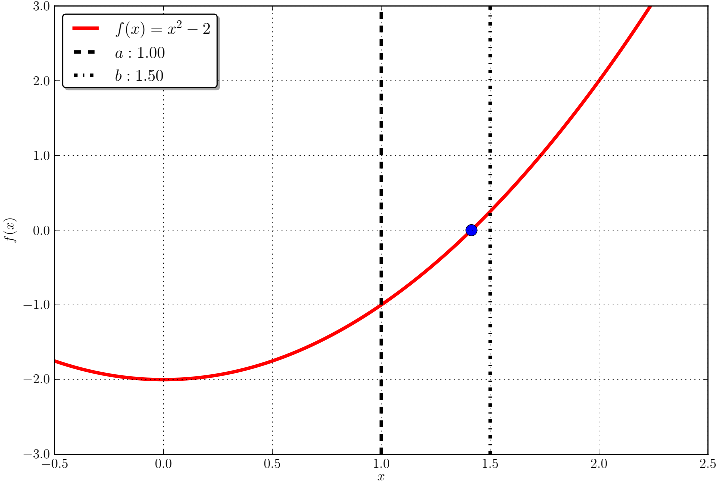
CSC 349a (3)



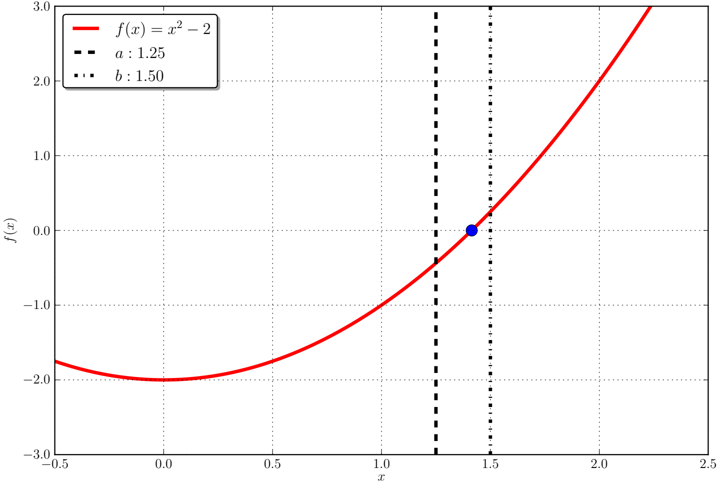
Observation: Since $0 \le \sqrt{2} \le 2$, the value of $\sqrt{2}$ can be approximated by finding a root of the polynomial $f(x) = x^2 - 2$ in the range $[a, b]$, where $a = 0$ and $b = 2$.



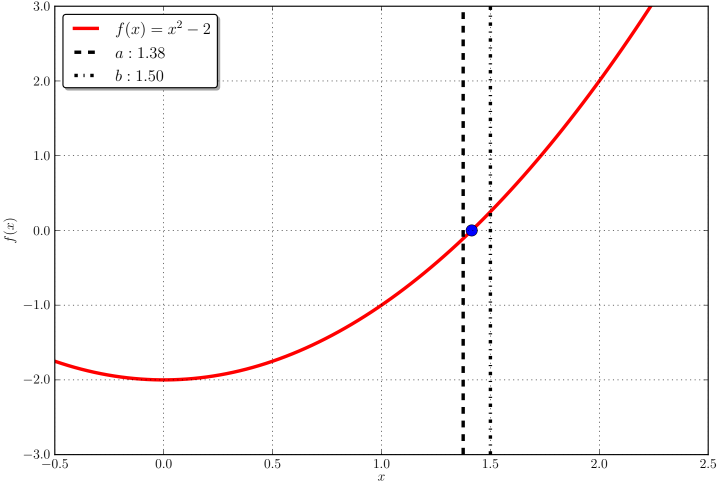
By iteratively refining the range $[a, b]$ until a and b converge, the value of $\sqrt{2}$ can be approximated to any number of significant digits.



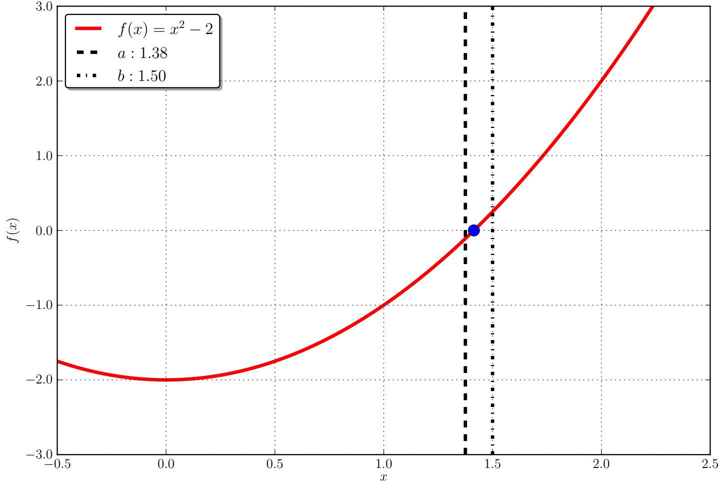
By iteratively refining the range $[a, b]$ until a and b converge, the value of $\sqrt{2}$ can be approximated to any number of significant digits.



This algorithm is called **bisection**.

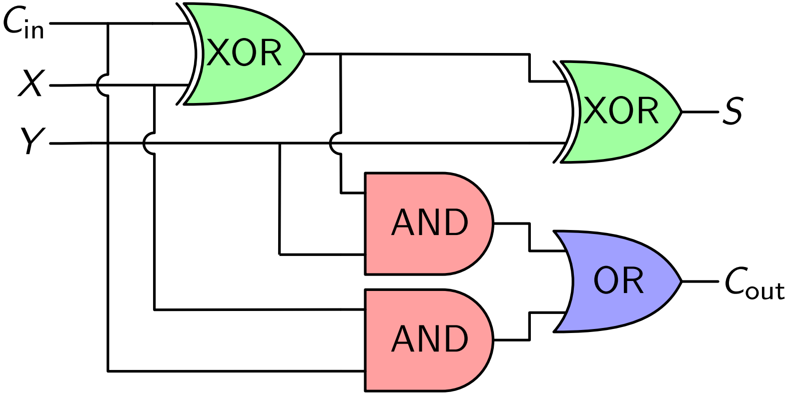


Question: If *n* significant digits of precision are required, what is the running time of the bisection algorithm? Does a faster algorithm exist?

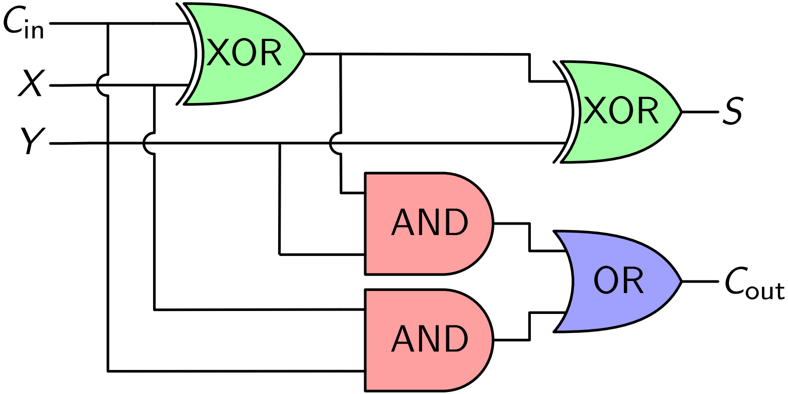


CSC 349a: Numerical Analysis

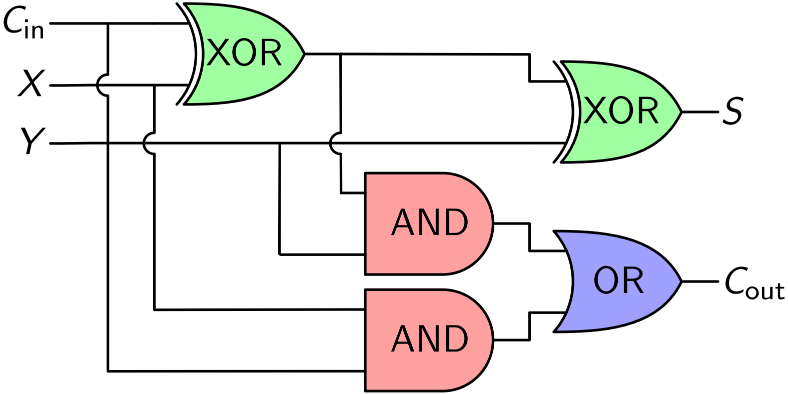
Floating point representation, rounding and truncation, root finding, interpolation, numerical linear algebra, numerical differentiation and integration.



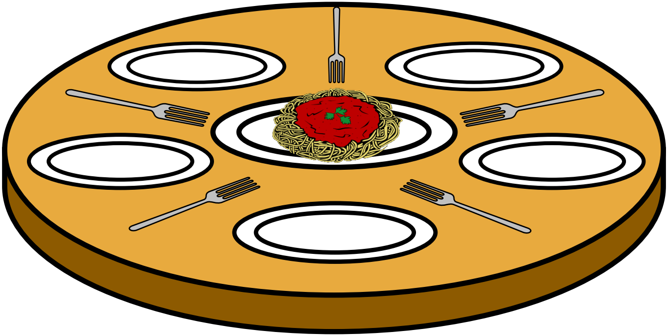
CSC 355: Digital Logic and Computer Organization
Circuit design, finite state machines, processor design.



CSC 355: Digital Logic and Computer Organization
A common CSC 355 assignment is to design an ALU out of logic gates.

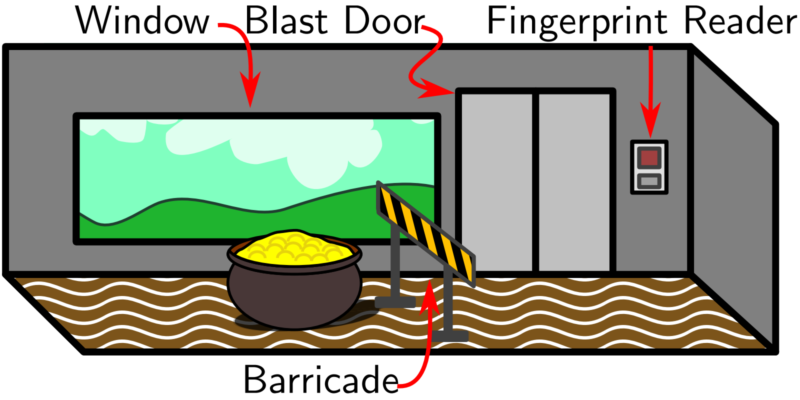


CSC 355: Digital Logic and Computer Organization
The diagram above shows a 1-bit full adder, which adds two bits X and Y, along with a carry-in bit C_{in}, to produce an output bit S and a carry-out bit C_{out}.



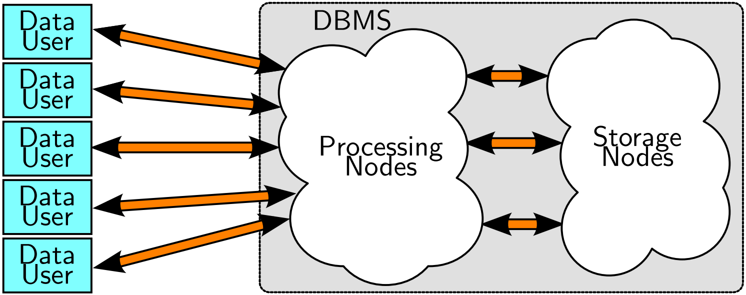
CSC 360: Operating Systems

Process creation and management, file systems, resource management, multithreaded programming.



SENG 360: Security Engineering

Access control schemes, malware, network security and intrusion detection, social engineering.



CSC 370: Database Systems

Database management systems, data processing, relational algebra, efficient data storage and retrieval.

400-level courses on algorithms and theory

CSC 421: Introduction to AI (or **ECE 470**: AI)

CSC 422: Graph Algorithms

CSC 423: Randomized Algorithms

CSC 425: Analysis of Algorithms

CSC 426: Computational Geometry

CSC 428a: Combinatorial Algorithms

CSC 429: Cryptography