Analysis of Algorithms

V. Adamchik

CSCI 570

Lecture 1

University of Southern California

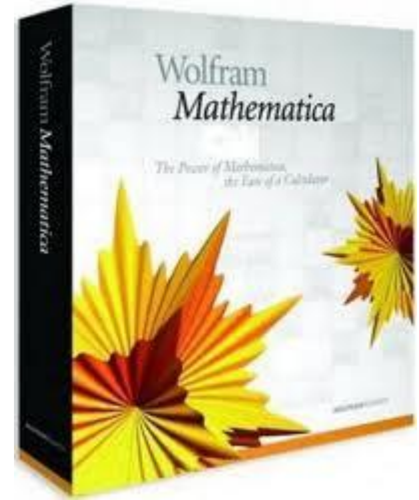
Spring 2023

Review

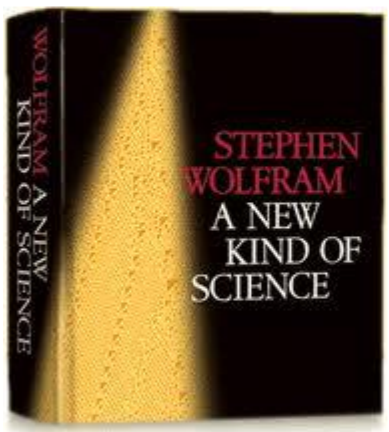
Reading: chapter 1

About Myself

1990-2010: *was working on Mathematica*



WOLFRAM
RESEARCH



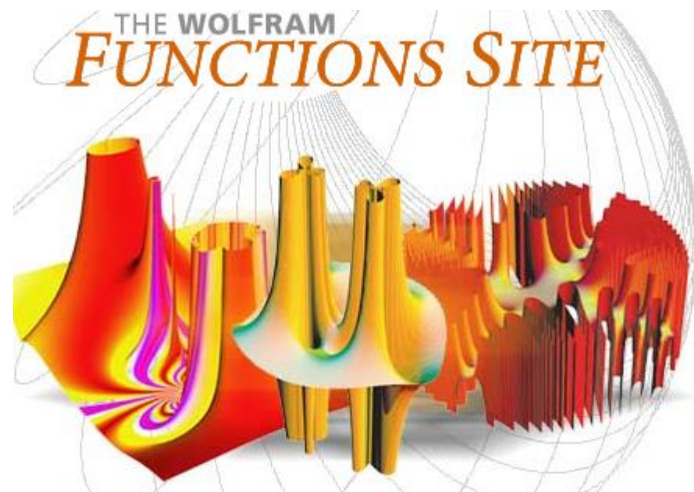
WolframAlpha™
computational™
knowledge engine

Wolfram *Mathematica*®
ONLINE INTEGRATOR
The world's only full-power integration solver

HOW TO ENTER INPUT | RANDOM EXAMPLE

$\int$ 1/(x^(1/4) + x^(1/3)) *dx*

Compute Online With *Mathematica*



About Myself

Carnegie Mellon
School of Computer Science

2000-2016: with Carnegie Mellon University, Pittsburgh

"Walking to the sky"

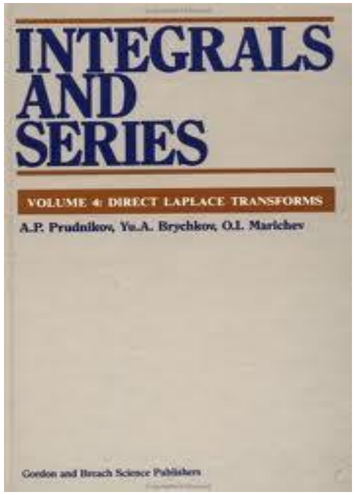
Bill Gates building for SCS

About Myself

Since 2016: University of Southern California, Los Angeles

https://viterbi-web.usc.edu/~adamchik/

Research



My area of research is computer algebra and symbolic computation.

Computer algebra is the field of math and cs that is concerned with development and implementation of algorithms that allow one to perform specific symbolic mathematical computations (like differentiation, integration …). It is spanning many different scientific and technical domains, like AI and ML.

I have published over 70 research articles

Chaitin's Omega constant (https://en.wikipedia.org/wiki/Omega_constant)

Bendersky-Adamchik's constants

Chapter 1.1: Runtime Complexity

The term analysis of algorithms is used to describe approaches to study the performance of computer programs. We interested to find a runtime complexity of a particular algorithm as a function of T(n) that describes a relation between algorithm's execution time and the input size n that tends to infinity:

$$\boxed{\lim_{n \to \infty} T(n)}$$

Consider a problem of addition of two n-bit binary numbers on 64-bit CPU. Let T(n) represent an amount of time used to add two n-bit numbers. If $n < 64$, addition is performed entirely by CPU. We say, it takes constant time. However, for large integers $n > 64$, those numbers do not fit CPU, so they have to be added in memory. In this case, the complexity of addition is a constant anymore, but a function of the input size.

Runtime Complexity

In this course we will perform the following types of analysis:

- the worst-case complexity
- the best-case complexity
- the average case complexity
- the amortized time complexity

We measure the runtime of an algorithm using following asymptotic notations: O, Ω, Θ.

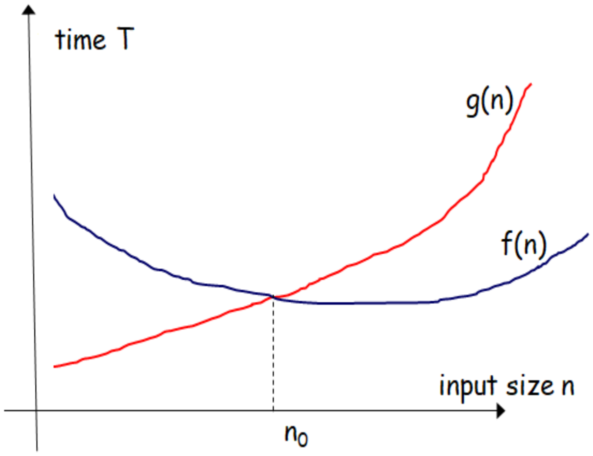
Big-O (upper bound)

For any monotonic functions f, g from the positive integers to the positive integers, we say

$$f(n) = O(\ g(n)\ )$$

if

g(n) eventually dominates f(n)



Formally: there exist constants c and n_0 such that for all *sufficiently large* n: $f(n) \leq c \cdot g(n)$

$$\exists c, n_0\ \forall n : n \geq n_0,\ f(n) \leq c \cdot g(n)$$

Example: $n^2 + 2n + 1 = O(n^2)$

$\exists c, n_0\ \forall n : n \geq n_0,\ f(n) \leq c \cdot g(n)$

Proof.

We need to find c and n_0.

Observe, that since $1 \leq n$, we get

$$n^2 + 2\,n + 1 \leq n^2 + 2\,n^2 + n^2 = 4\,n^2 \text{ for } n \geq 1.$$

We choose $c = 4$ and $n_0 = 1$.

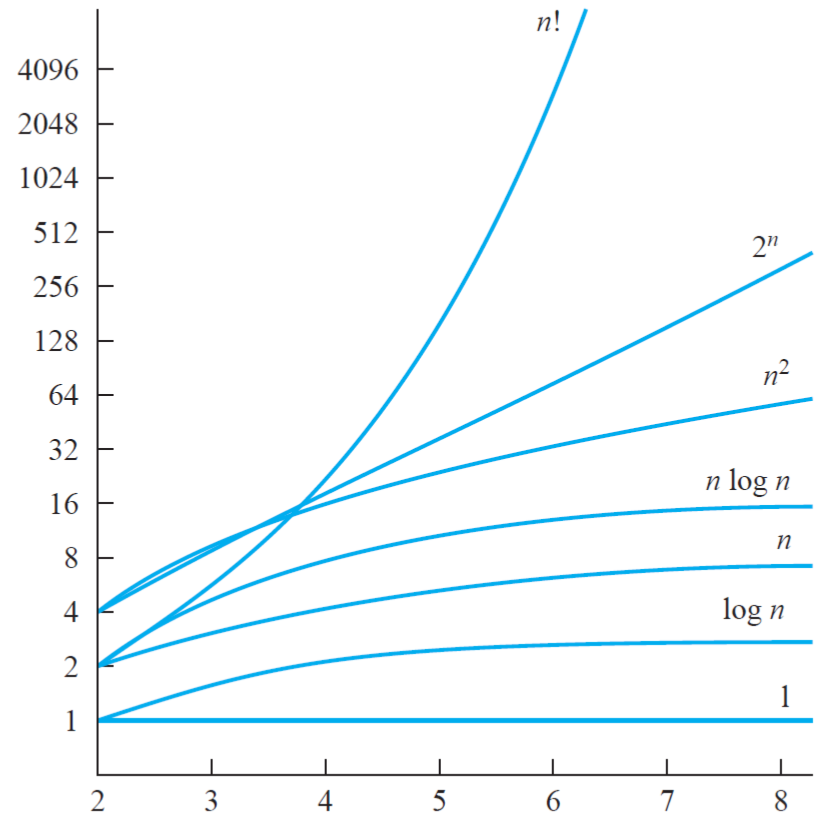
Hierarchies of Functions

Polynomial functions grow faster than logarithmic functions

$\log n = O(n)$
$\log^2 n = O(n)$
$\log^{1000} n = O(n)$

Exponential functions grow faster than polynomial ones

$n = O(2^n)$
$n^{10} = O(2^n)$
$n^{1000} = O(2^n)$



Discussion Problem 1

Arrange the following functions (no formal proof required) in increasing order of growth rate with g(n) following f(n) in your list if and only if *f(n) = O(g(n))*

$$\log n^n,\ n^2,\ n^{\log n},\ n \log\log n,\ n^{1/\log n}\ \ 2^{\log n},\ \log^2 n,\ n^{\sqrt{2}}$$

Discussion Problem 2

Suppose that *f(n)* and *g(n)* are two positive non-decreasing functions such that $f(n) = O(g(n))$. Is it true that $2^{f(n)} = O(2^{g(n)})$?

Omega: Ω (lower bound)

For any monotonic functions f, g from the positive integers to the positive integers, we say

$$f(n) = \Omega(\, g(n)\,)$$

if:

f(n) eventually dominates g(n)

Formally: there exist constants c and n_0 such that for all *sufficiently large* n: $f(n) \geq c \cdot g(n)$

$$\exists c, n_0 \ \forall n : n \geq n_0,\ f(n) \geq c \cdot g(n)$$

Example: $n^2 + 2n + 1 = \Omega(n^2)$

$\exists c, n_0 \; \forall n : n \geq n_0, \; f(n) \geq c \cdot g(n)$

Proof.

We need to find c and n_0.

Observe, that

$$n^2 + 2\,n + 1 \geq n^2 \text{ for } n \geq 1.$$

We choose $c = 1$ and $n_0 = 1$.

Discussion Problem 3

Suppose that $f(n)$ and $g(n)$ are two positive non-decreasing functions such that $f(n) = \Omega(g(n))$. Is it true that $2^{f(n)} = \Omega(2^{g(n)})$?

Theta: Θ

For any monotonic functions f, g from the positive integers to the positive integers, we say

$$f(n) = \Theta(\ g(n)\ )$$

if:

$$f(n) = O(\ g(n)\ ) \quad \underline{\text{and}} \quad f(n) = \Omega(\ g(n)\ )$$

Formally:

$$\exists\ c_1, c_2, n_0\ \forall n : n \geq n_0,\ c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$

In this class we will be mostly concerned with a big-O notation.

Quickies

1. $n = \Omega(n^2)$?
2. $n = \Theta(n + \log n)$?
3. $\log n = \Omega(n)$?
4. $n^2 = \Omega(n \log n)$?
5. $n^2 \log n = \Theta(n^2)$?
6. $3n^2 + 4n + 5 = \Theta(n^2)$?
7. $2^n + 100n^2 + n^{100} = \Omega(n^{101})$?
8. $(1/3)^n + 100 = \Theta(1)$?

Discussion Problem 4

Consider the following pseudocode. What is the Big-O runtime complexity of the following function?

```
int FindMax(int[] A, int len) {
   int max = A[0];
   for (int i = 1; i < len; i++) {
        if (max < A[i]) max = A[i];
   }
   return max;
}
```

Among all operations in this code snippet, define the most expensive operation, and then count how many times it's executed as a function of the input size.

Discussion Problem 5

Consider the following pseudocode. What is the Big-O runtime complexity of the following function?

```
void bigOh1(int n) {
    for (int i = 1; i < len; i++) {
      int j=1;
       while (j < i)
              j = j*2;
    }
}
```

Discussion Problem 6

What is the Big-O runtime complexity of the following function? Give the tightest bound.

```
void bigOh2(int[] A, int n)
    while (n > 0)
        find_max(A, n); //finds the max in A[0…n-1]
        n = n/4;
```

Discussion Problem 7

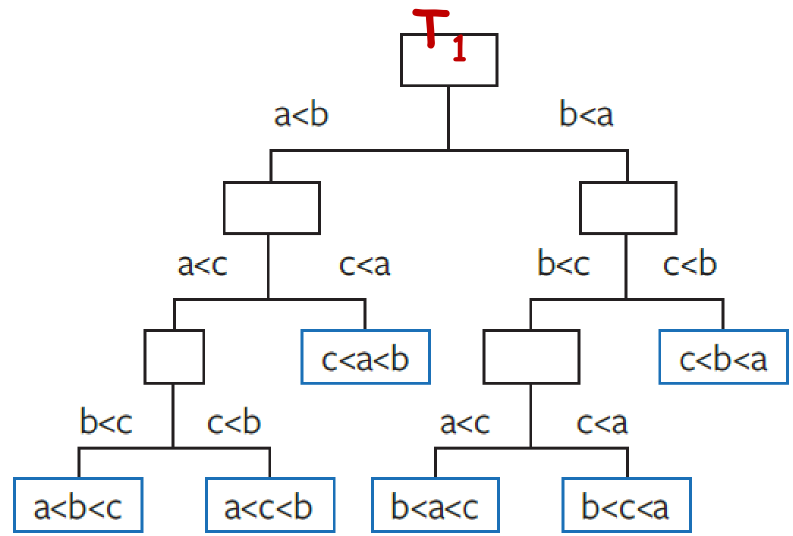
What is the Big-O runtime complexity of the following function?

```
string bigOh3(int n) {
    if(n == 0) return "a";
    string str = bigOh3(n-1);
    return str + str; /*concatenation */
}
```

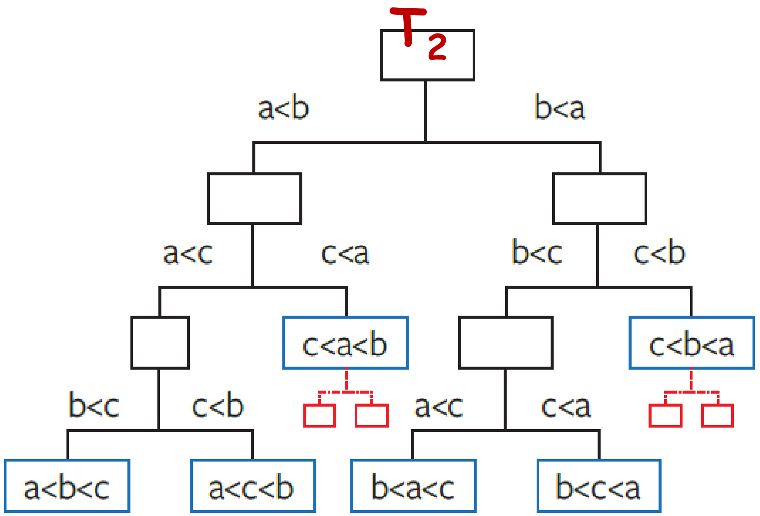
Chapter 1.2: Sorting Lower Bound

We will show here that any deterministic comparison-based sorting algorithm must take $\Omega(n \log n)$ time to sort an array of n elements in the worst-case.

As an example, consider an array of three numbers $a \neq b \neq c$. We will sort them by comparisons.



A tree with six leaves



A tree with eight leaves

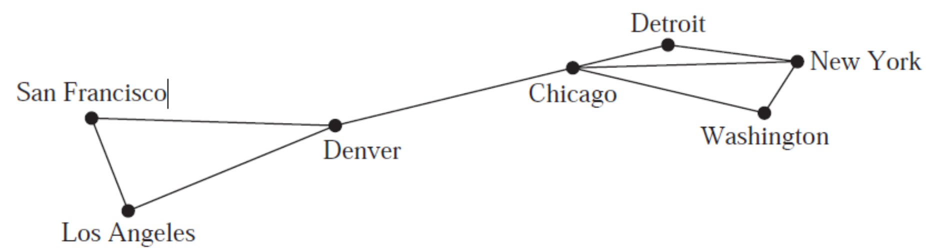
Chapter 1.3: Trees and Graphs

A graph G is a pair (V, E) where V is a set of vertices (or nodes) E is a set of edges connecting the vertices.
An undirected graph is connected when there is a path between every pair of vertices.
A graph is simple if it has no self-loops and no multi-edges.
A path in a graph is a sequence of distinct vertices.
A cycle is a path that starts and ends at the same vertex.
A tree is a connected graph with no cycles.

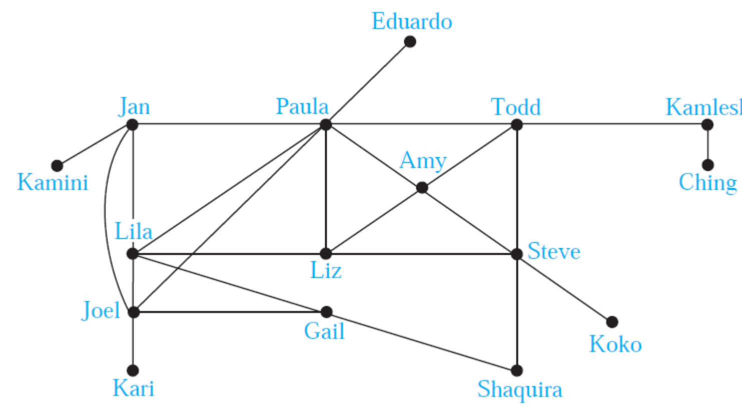
We start with reviewing mathematical proofs (induction and contradiction).

Graph Models

A computer network can be modeled using a graph in which the vertices of the graph represent the data centers and the edges represent communication links.



A social network can be modeled using a graph in which individuals or organizations are represented by vertices; relationships between individuals or organizations are represented by edges.



The WorldWideWeb can be modeled as a directed graph where each Web page is represented by a vertex and where an edge starts at the Web page *a* and ends at the Web page *b* if there is a link on *a* pointing to *b*.

We can use graphs to model many different types of transportation networks, including road, air, and rail networks, as well shipping networks.

Theorem. Let G be a graph with V vertices and E edges. The following statements are equivalent:

1. G is a tree (a connected graph with no cycles).
2. Every two vertices of G are connected by a unique path.
3. G is connected and $V = E + 1$.
4. G is acyclic and $V = E + 1$.
5. G is acyclic and if any two non-adjacent vertices are joined by an edge, the resulting graph has exactly one cycle.

$1 \Rightarrow 2$

1. G is a tree.

2. Every two nodes of G are joined by a unique path.

Proof:

$2 \Rightarrow 3$

2. Every two nodes of G are joined by a unique path.

3. G is connected and $V = E + 1$.

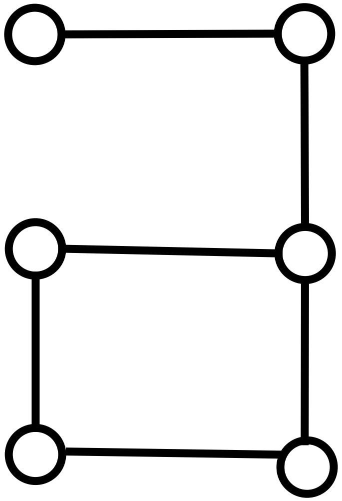
Proof:

$3 \Rightarrow 4$

3. G is connected and $V = E + 1$.

4. G is acyclic and $V = E + 1$.

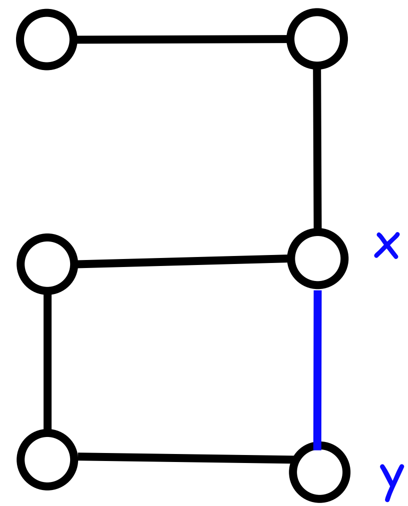
Proof: (by contradiction)



$4 \Rightarrow 5$

4. G is acyclic and $V = E + 1$.

5. G is acyclic and if any two non-adjacent nodes are joined by an edge, the resulting graph has exactly one cycle.



Proof:

Since G is connected, there is a path from x to y.

Let $G' = G + (x, y)$

Then G' will have a single cycle created by a path from x to y and that new edge (x, y).

Representing Graphs

Adjacency List

or

Adjacency Matrix

Vertex X is *adjacent* to vertex Y if and only if there is an edge (X, Y) between them.

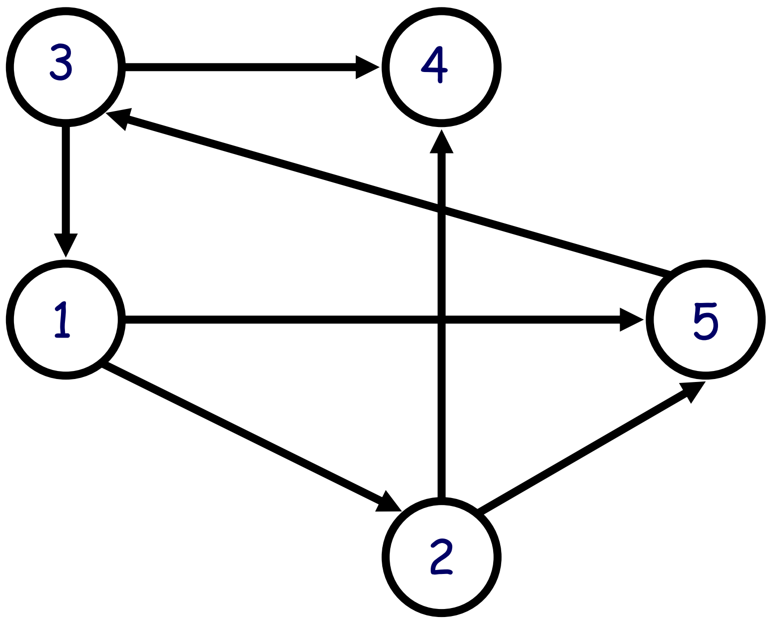
Adjacency List Representation is used for representation of the sparse ($E = O(V)$) graphs.

Adjacency Matrix Representation is used for representation of the dense ($E = \Omega(V^2)$) graphs.

Theorem. Prove that in an undirected simple graph $G = (V, E)$, there are at most $V(V-1)/2$ edges.

In short, using the asymptotic notation, $E = O(V^2)$.

Adjacency Matrix Representation



If there is an edge (u, v), we put 1 into the table (matrix), o.w. it is 0.

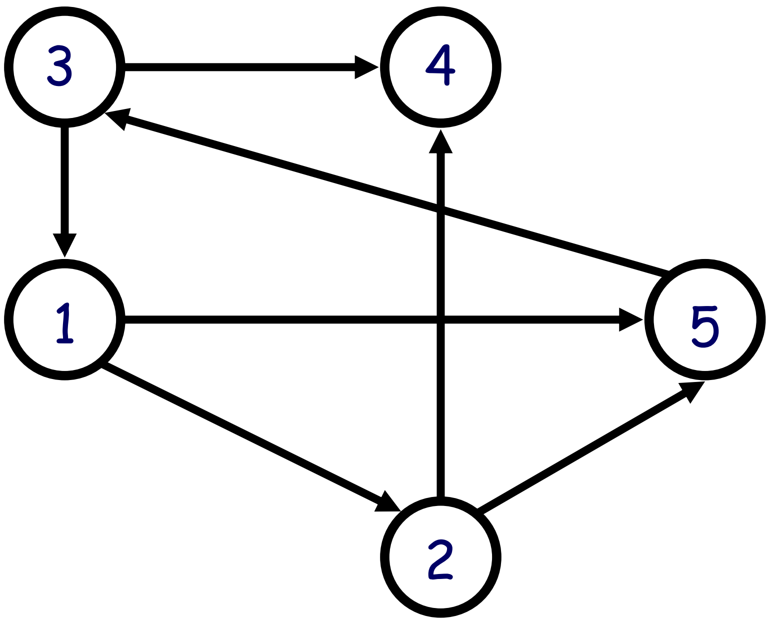
$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

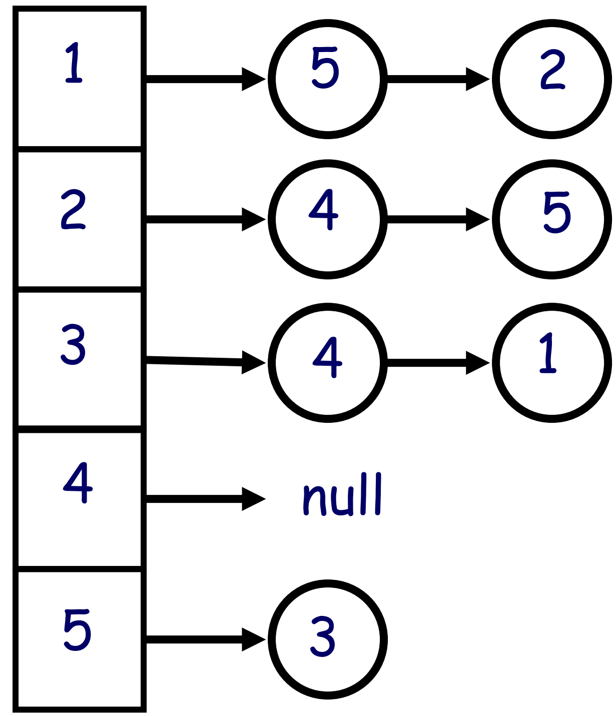
The adjacency matrix representation requires a lot of space: for a graph with V vertices, we must allocate space in $O(V^2)$.

Is vertex 1 adjacent to 3?

It takes constant time to figure it out.

Adjacency List Representation





In an adjacency list representation, we have a one-dimensional array of vertices, where each vertex contains a linked list of all the other vertices connected to that vertex.

Adjacency lists require O(V + E) space.

Is vertex 1 adjacent to 3?

It takes linear time to figure it out.

Graph Traversals

Graph traversal (also known as graph search) refers to the process of visiting each vertex in a graph. The traversal may require that some vertices be visited more than once. Since a graph is a nonlinear data structure, there is no unique traversal. There are two algorithms:

Depth-First-Search (DFS): It starts at a selected node and explores as far as possible along each branch before backtracking. DFS uses a stack for backtracking.

Breadth-First-Search (BFS): It starts at a selected node and explores all nodes at the present depth prior to moving on to the nodes at the next depth level. BFS uses a queue for bookkeeping.

Runtime complexity: $O(V + E)$

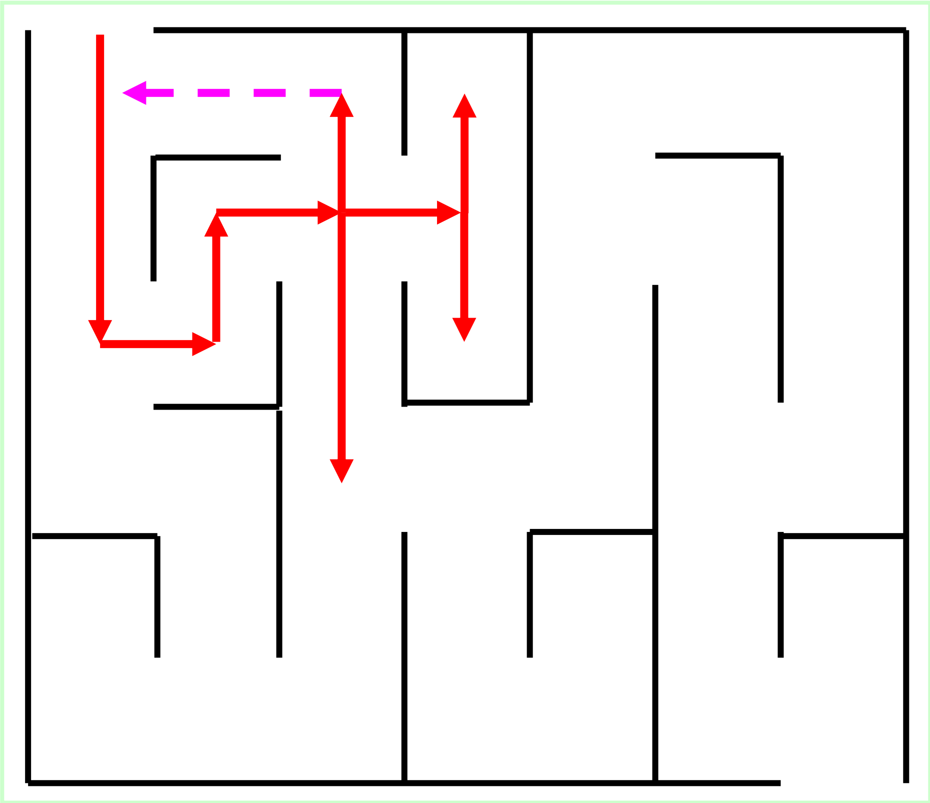
Property 1: They visit all the vertices in the connected component.

Property 2: The result of traversal is a spanning tree of the connected component.

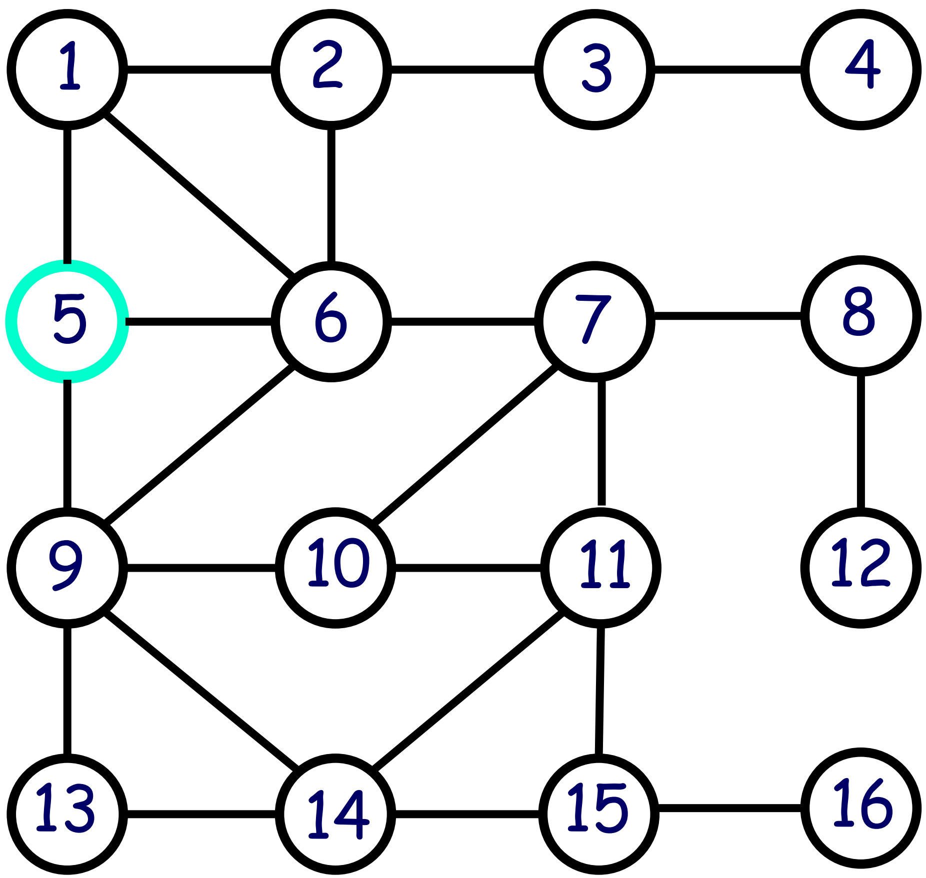
DFS and Maze Traversal

The DFS algorithm is similar to a classic strategy for exploring a maze

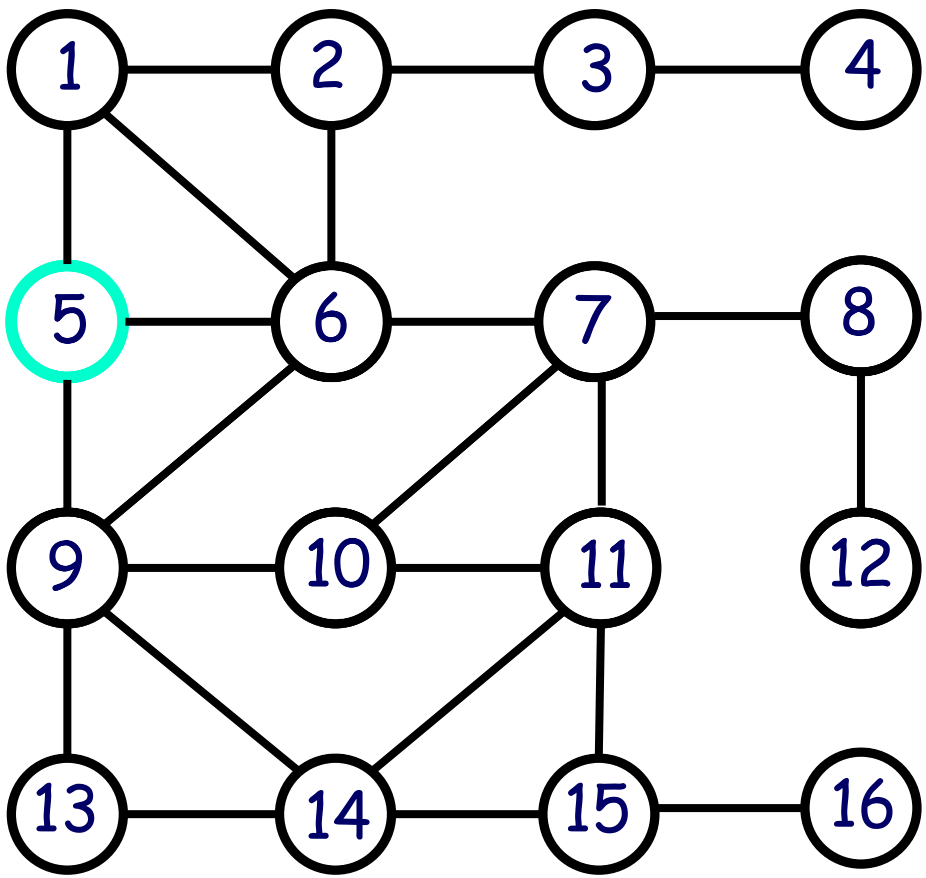
- We mark each intersection, corner and dead end (vertex) visited
- We mark each corridor (edge) traversed
- We keep track of the path back to the entrance (start vertex) by means of a rope.



Perform a DFS on the following graph



Perform a BFS on the following graph



Discussion Problem 8

The complete graph on n vertices, denoted K_n, is a simple graph in which there is an edge between every pair of distinct vertices.

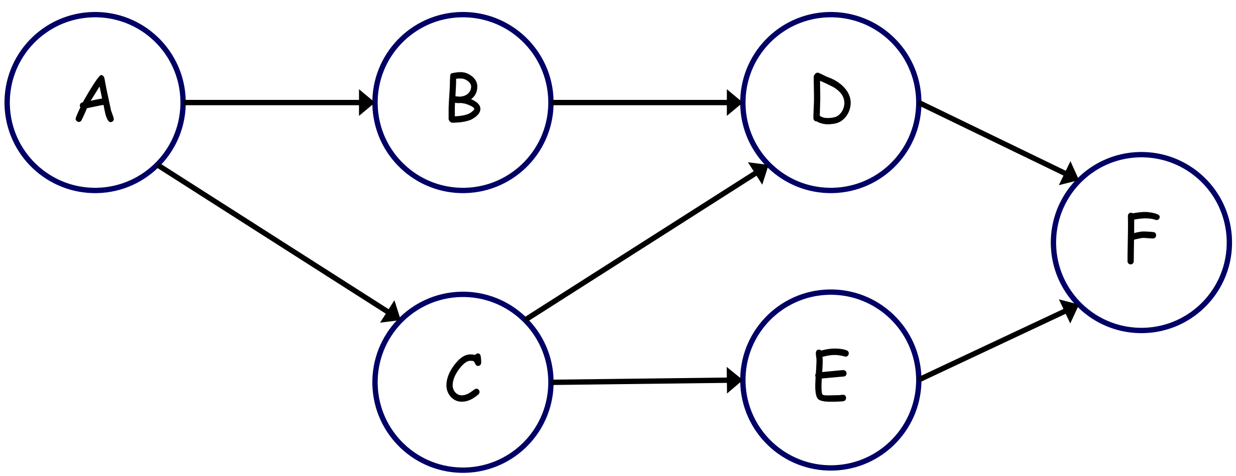
What is the height of the DFS tree for the complete graph K_n?

What is the height of the BFS tree for the complete graph K_n?

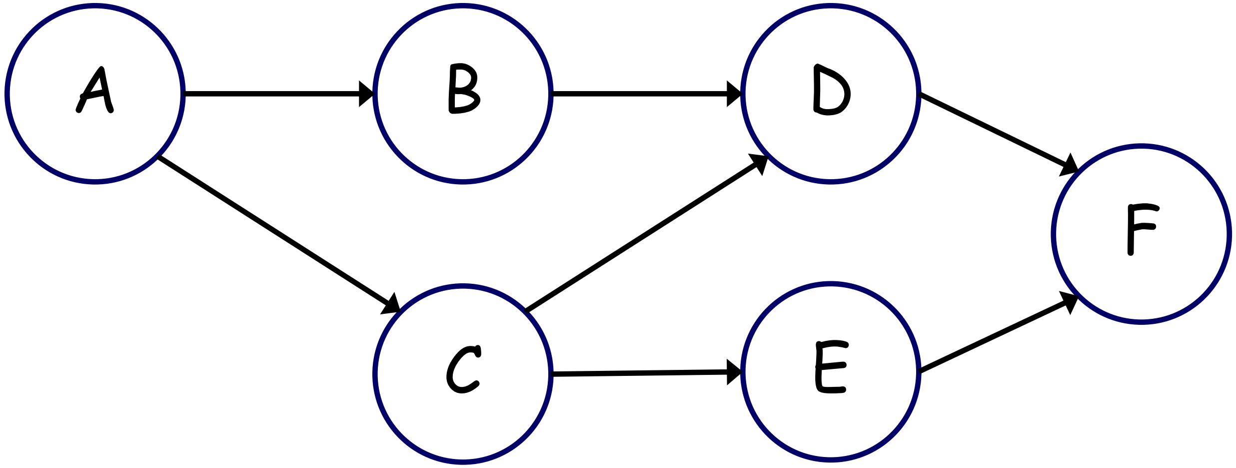
Topological Sort for DAG

Suppose each vertex represents a task that must be completed, and a directed edge (u, v) indicates that task v depends on task u. That is task u must be completed before v. The topological ordering of the vertices is a valid order in which you can complete the tasks.

DAG = Directed Acyclic Graph.

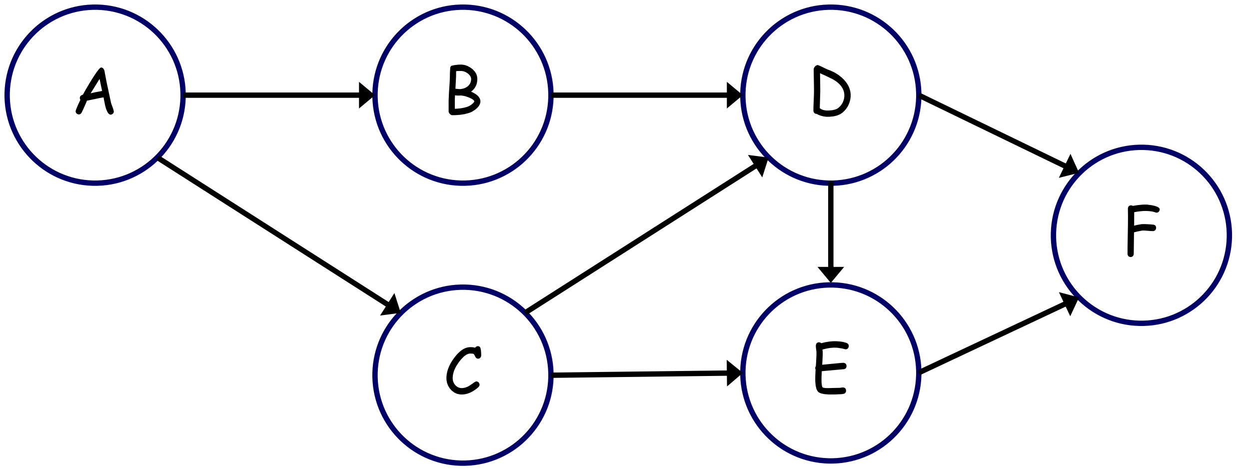


How to find a topological order?



1. Select a vertex that has zero in-degree.
2. Add the vertex to the output.
3. Delete this vertex and all its outgoing edges.
4. Repeat.

Linear Time Algorithm



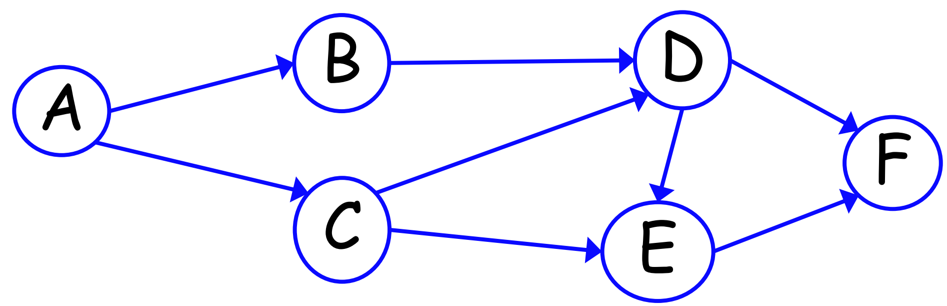
1. Select a vertex.
2. Run DFS and return vertices that has no undiscovered leaving edges.
3. You may run DFS several times.

You get vertices in reverse order.

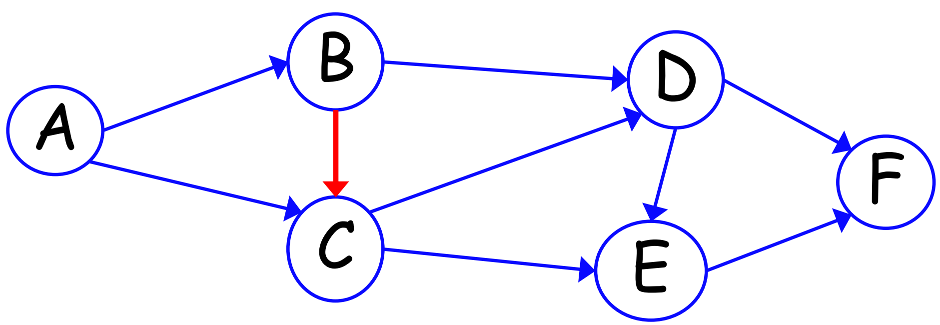
Why is it a linear time algorithm?

Discussion Problem 9

Suppose instead we are interested in finding the *longest* path in a directed acyclic graph (DAG). In particular, we are interested in a path that visits all vertices. Give a linear-time algorithm to determine if such a path exist.



Graph G_1



Graph G_2

Ch1: exercises

4. Arrange the following functions

$$4^{\log n}, \sqrt{\log n}, n^{\log\log n}, (\sqrt{2})^{\log n}, 2^{\sqrt{2\log n}}, n^{1/\log n}, (\log n)!$$

in increasing order of growth rate with $g(n)$ following $f(n)$ in your list if and only if $f(n) = O(g(n))$.