ASYMPTOTIC NOTATION AND DATA STRUCTURES

CS 6212 – Design and Analysis of Algorithms

LOGISTICS

- **Instructor**

  Prof. Amrinder Arora

  amrinder@gwu.edu

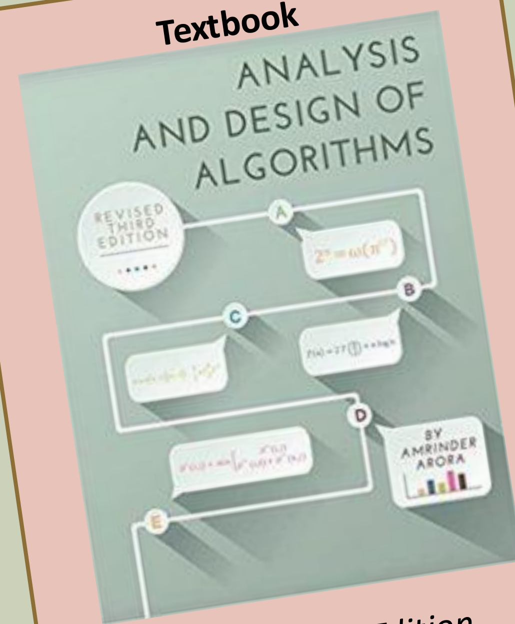
  Please copy TA on emails

  Please feel free to call as well

  ☺

- Available for study sessions
  Science and Engineering Hall
  GWU

Textbook



Revised Third Edition
Cognella Academic Publisher
www.cognella.com

ISBN: **978-1793549952**

RECAP

- Asymptotic Notation
  - Big Oh
  - Small Oh
  - Big Omega
  - Small Omega
  - Theta

ASYMPTOTIC NOTATIONS

- Big O notation
  - $f(n) = O(g(n))$ if there exist constants n_0 and c such that $f(n) \leq c\, g(n)$ for all $n \geq n_0$.

  For example, $n = O(2n)$ and $2n = O(n)$

  If $f(n) = a_0 n^0 + a_1 n^1 + \ldots + a_m n^m$,
  then $f(n) = O(n^m)$
- Big Omega notation
  - $f(n) = \Omega(g(n))$ if there exist constants n_0 and c such that $f(n) \geq c\, g(n)$ for all $n \geq n_0$.
- Small o notation
  - $f(n) = o(g(n))$ if for any constant $c > 0$, there exists n_0 such that $0 \leq f(n) < c\, g(n)$ for all $n \geq n_0$.

  For example, $n = o(n^2)$
- Small omega (ω) notation
  - $f(n) = \omega(g(n))$ if for any constant $c > 0$, there exists n_0 such that $f(n) \geq c\, g(n) \geq 0$, for all $n \geq n_0$

  For example, $n^3 = \omega(n^2)$
- Theta (Θ or θ) notation
  - If $f(n) = O(g(n))$ and $g(n) = O(f(n))$, then $f(n) = \Theta(g(n))$

ASYMPTOTIC NOTATIONS (CONT.)

- Transpose symmetry
  - $f(n) = O(g(n))$ if and only if $g(n) = \Omega(f(n))$
  - $f(n) = o(g(n))$ if and only if $g(n) = \omega(f(n))$

- Limit method
  - $f(n) = o(g(n))$ implies $\lim_{n\to\infty} f(n)/g(n) = 0$
  - Using L'Hopital's rule is common when using this method.

ASYMPTOTIC NOTATIONS (CONT.)

Analogy with real numbers

O	o	Θ	ω	Ω
$\leq$	$<$	$=$	$>$	$\geq$

ASYMPTOTIC NOTATIONS (CONT.)

Which properties apply to which (of 5) asymptotic notations?

- Transitivity
- Reflexivity
- Symmetry
- Transpose Symmetry
- Trichotomy

ASYMPTOTIC NOTATIONS (CONT.)

Which properties apply to which (of 5) asymptotic notations?

- Transitivity: O, o, Θ, ω, Ω
- Reflexivity: O, Θ, Ω
- Symmetry: Θ
- Transpose Symmetry: (O with Ω, o with ω)
- Trichotomy: Does not hold. For real numbers x and y, we can always say that either x < y or x = y or x > y. For functions, we may not be able to say that. For example if f(n) = sin(n) and g(n)=cos(n)

ASYMPTOTIC NOTATIONS (CONT.)

Analogy with real numbers

O	o	Θ	ω	Ω
≤	<	=	>	≥

Question: Does it still not hold if we limit ourselves to functions that are positive, always increasing with n, and are not trigonometric?

ASYMPTOTIC NOTATIONS (CONT.)

Special Functions

- Polynomial vs. exponential
- Polynomial vs. logs
- Factorial / Combinatorial
- Trigonometric Functions
- Floors and Ceilings

How do we prove that 2^n = omega (n^k)?

We want to prove that for any given c, there exists n_0, such that 2^n > c * n^k, for all n > n_0.

DESIGNING AN ALGORITHM – TECHNIQUES

- Divide and Conquer
- Greedy Method
- Dynamic Programming
- Graph search methods
- Backtracking
- Branch and bound

HOW TO DESIGN A FAST ALGORITHM?

- Define the problem
- Find a working solution
- Fast enough?
- If not, you may have two options:
  - Consider a different technique
  - Consider a different data structure
- Iterate until satisfied.

DATA STRUCTURES

- A data structure is a structure to hold the data, that allows several interesting operations to be performed on the data set.
- **The data structure is designed with those specific operations in mind.**
- General problem:
  - Given a data set and the operations that need to be supported, come up with a data structure (organization) that allows those operations to be done in an efficient manner.

STACK

- Last In First Out (LIFO)
- Allows 3 operations:
  - Push (a)
  - Pop()
  - Top()

IMPLEMENTATION OF STACKS

- Using an array
  - Use an array S[1:N], and use a special pointer to the "top" of the stack.
  - When pushing something on the stack, increment the pointer
  - When popping, decrement the pointer
- Using a linked list
  - Use a special pointer to the "top" of the stack
  - When pushing something on the stack, advance the "top" pointer
  - When popping, move the "top" pointer back one step – this suggests that the linked list must be a doubly linked list

QUEUE

- First In First Out (FIFO)
- Allows 2 operations:
  - dequeue(): Returns the first element
  - enqueue(a): Adds an element a to the end of the queue

QUEUE – IMPLEMENTATION

tail -> …… -> head

- Using an array
  - Keep “head” and “tail” indexes
- Using a linked list
  - Keep “head” and “tail” pointers
- Handling operations
  - When enqueuing an item, move tail one step to the left.
  - When dequeuing an item, move head one step to the left

RECORD STRUCT OBJECT CLASS TEMPLATE

- A record is a built-in structure data type, that allows the packaging of several elements (called fields)
- Every high level language allows the user to define customized records.
  - In C#/Java, this is called “class”.
  - In C, this is called “struct”.

LINKED LISTS

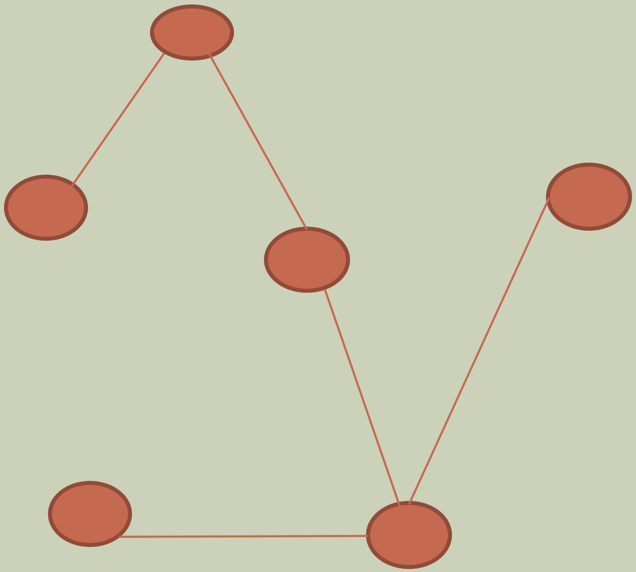
- Singly Linked
  - A singly linked list is a sequence of records, where every record has a field that points to the next record
  - A special pointer called “first” has the reference to the first record
- Doubly Linked
  - A doubly linked list is a sequence of records, where every record has a field that points to the next record, and a field that points to the previous record
  - Special pointers called “first” and “last” with references to the first and the last records

BASIC CONTENTION

- Array vs. List
  - Modify: Array does not allow structural changes
  - Access a random element: Array allows random element access

GRAPH

- A graph G=(V,E) consists of a finite set V, which is the set of vertices, and set E, which is the set of edges. Each edge in E connects two vertices v1 and v2, which are in V.
- Can be directed or undirected



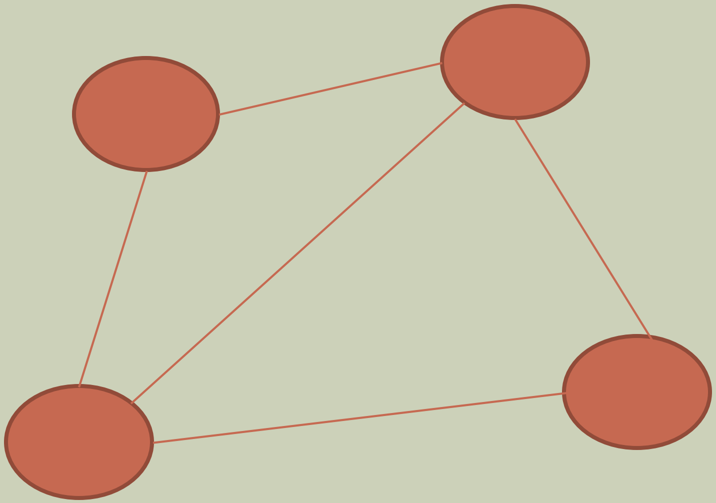
An example graph with n = 6, m = 5

GRAPH DEFINITIONS

- If (x,y) is an edge, then x is said to be adjacent to y, and y is adjacent from x.
- In the case of undirected graphs, if (x,y) is an edge, we just say that x and y are adjacent (or x is adjacent to y, or y is adjacent to x). Also, we say that x is the neighbor of y.
- The indegree of a node x is the number of nodes adjacent to x
- The outdegree of a node x is the number of nodes adjacent from x
- The degree of a node x in an undirected graph is the number of neighbors of x
- A path from a node x to a node y in a graph is a sequence of node x, $x_1, x_2, \ldots, x_n$, y, such that x is adjacent to x_1, x_1 is adjacent to x_2, ..., and x_n is adjacent to y.
- The length of a path is the number of its edges.
- A cycle is a path that begins and ends at the same node
- The distance from node x to node y is the **length of the shortest path** from x to y.

A GRAPH WITH N = 4, M = 5

- Vertices, Edges and Faces (n, m, f)
  - n = 4, m = 5, f = 3
- This graph is planar (the graph can be laid out on a plane such that the edges don't cross each other)

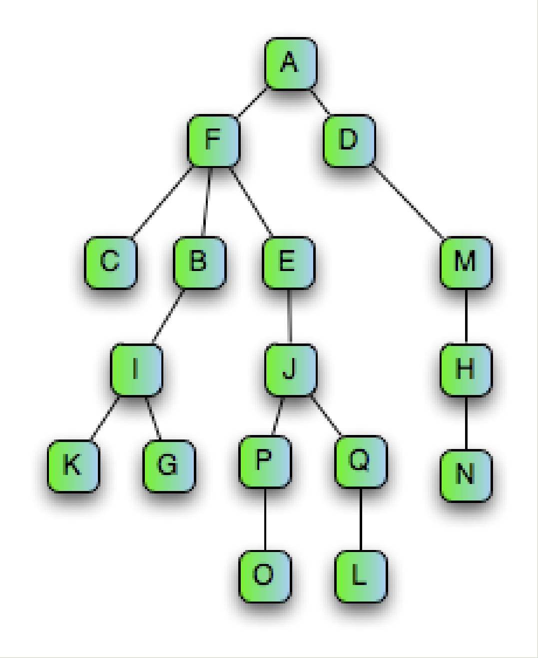


GRAPH REPRESENTATIONS

- Using a matrix A[1..n,1..n] where A[i,j] = 1 if (i,j) is an edge, and is 0 otherwise. This representation is called the adjacency matrix representation. If the graph is undirected, then the adjacency matrix is symmetric about the main diagonal.
- Using an array Adj[1..n] of pointers, which Adj[i] is a linked list of nodes which are adjacent to i.
- The matrix representation requires more memory, since it has a matrix cell for each *possible* edge, whether that edge exists or not. In adjacency list representation, the space used is directly proportional to the number of edges.
- If the graph is sparse (very few edges), then adjacency list may be a more efficient choice.

TREE

- A tree is a connected acyclic graph (i.e., it has no cycles)
- Rooted tree: A tree in which one node is designated as a root (the top node)



Example:
Node A is root node
F and D are child nodes of A.
P and Q are child nodes of J.
Etc.

TREE

- Definitions
  - Leaf is a node that has no children
  - Ancestors of a node x are all the nodes on the path from x to the root, including x and the root
  - Subtree rooted at x is the tree consisting of x, its children and their children, and so on and so forth all the way down
  - Height of a tree is the maximum distance from the root to any node

BINARY TREE

- A tree where every node has at most two children
- Binary Search Tree (BST): BST is a binary search tree where every node contains a value, and for every node x, all the nodes of the left subtree of x have values <= x, and all nodes in the right subtree of x have values >= x.
- BST supports 3 operations: insert(x), delete(x) and search(x)
- It is more interesting (and efficient) if the BST is "height balanced". Red Black and AVL trees are interesting implementations of height balanced BSTs.

WHY BSTS ARE OF INTEREST

- Array
  - Search the array in O(log n) time. Sorted. Search, using binary search.
  - Modify the array (add or delete) → O(n) time
- Linked List
  - Add or delete in O(1) time
  - Search, will take O(n) time
- BSTs
  - Add, delete, search, all in O(log n) time
- 1 million operations, assume on average, n = 1 million
    - 30% are inserts/deletes/modifies, and 70% are searches
    - How much time, does an array take?
    - Linked List:
    - BST:

HEAPS

- Also known as priority queues
- Very efficient data structure to enforce priority, although do not enforce complete sorting
- Can be max heap or min heap
- Commonly represented using a heap tree (although, can also be a forest)

BTREE, 2-3 TREE

- Flexible data structure, where a node has a variable number of children (say between 2 and 4, both including, or between 50 and 100 both including)
- This variable number allows us to leave some "holes" in the tree to fill as insertions happen, thereby allowing insertions without changing the structure of the tree entirely.
- The variable number also allows us to treat deletions without changing the structure.

- 2-3 tree is a specific kind of BTree where each node can have 2 or 3 children.

http://www.slideshare.net/amrinderarora/btrees-great-alternative-to-red-black-avl-and-other-bsts

MOTIVATION OF BTREES

[In a tree, the number of leaf nodes are b^h. (Branching factor ^ height)]

- Motivation 1: File System (DB) behaves differently from RAM
  - Consider a scenario where you have 17 million records. In a binary tree, the height would be log_2(17 million), that is, 25.
  - A 25 height BST in the main memory / RAM is no problem at all.
    - 25 x 1 nsec (assuming a slow 1 GHz processor).
  - But, in database, we would need to go to 25 ”locations”
    - 25 x 100 msec would be catastrophic (2.5 seconds!)
  - For this reason, a Btree has a branching factor of 50/100 as needed as opposed to BST's branching factor (2).

- Motivation 2: Rearrangement
  - Rearrangements in File System are very bad
  - So, you need flexibility, and gaps.

UNION FIND

- Also called “Disjoint Set” data structure
- How to maintain sets dynamically – sets can be merged (union), and we want to see which set a particular element is in.

- find(x) → Identifies the set that element x belongs to
- Union (S1, S2) → Combines these two sets

DISJOINT SETS WITHOUT UNION FIND

- Array
  - A[i] → Group Name
- Merge (G1, G2) → G1
  - Iterate the entire array
  - Wherever you see G2, call it G1
- Step Complexities
  - Find → O(1)
  - Merge → O(n)

  - N finds and m Merges → mn,   n^2

UNION FIND DATA STRUCTURE

- Each set is marked by a leader
- When calling “find” on a set’s member, it returns the leader
- Leader maintains a rank (or height)
- When doing a union, make the tree with smaller height (or rank) to be a child of the tree with the larger height
- Note that this is NOT a binary tree.

UNION FIND – PATH COMPRESSION

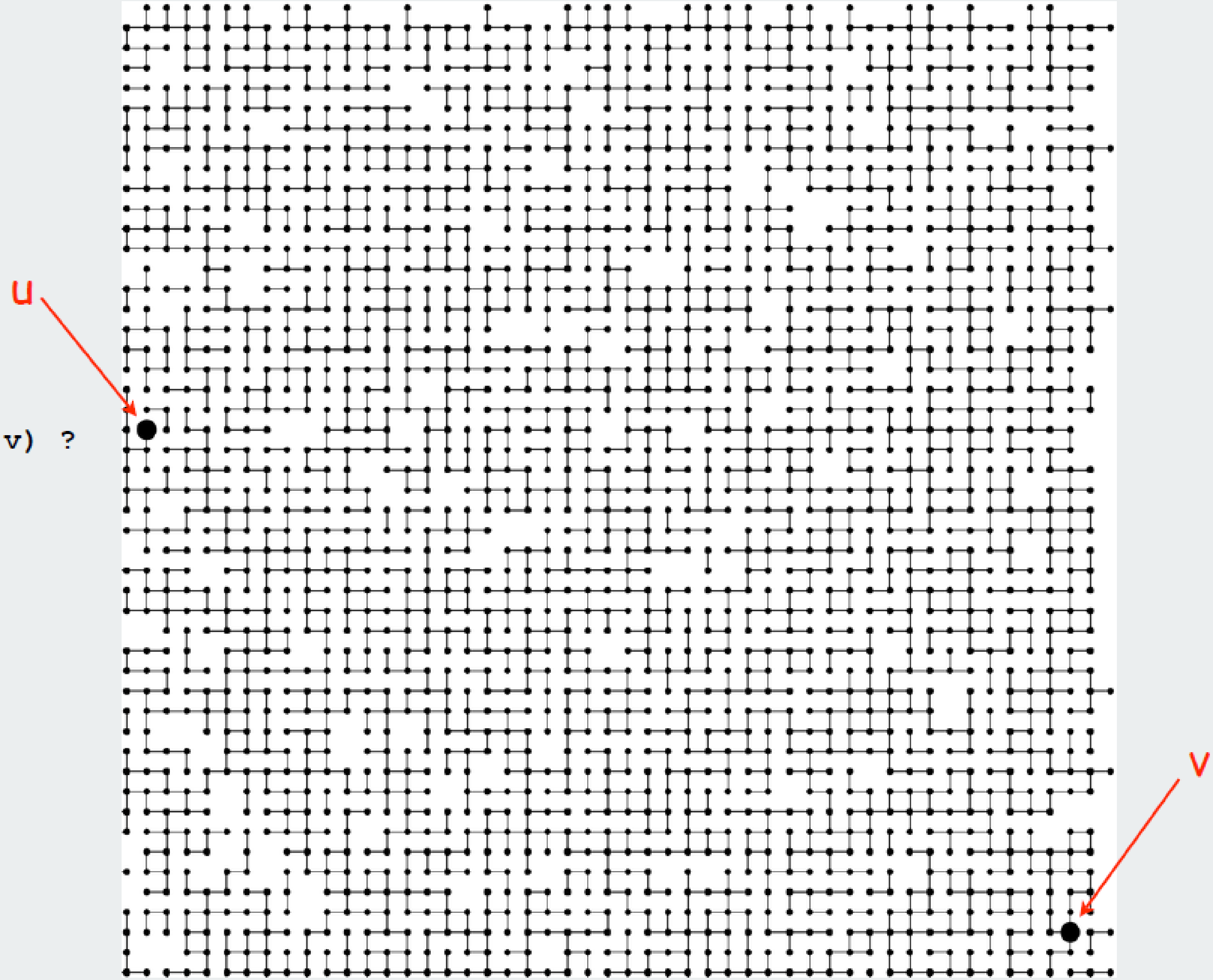
- When doing a find, follow that up by compressing the path to the root, by making every node (along the way) point to the root.
- This is not easy to prove, but Union Find with Path compression, when starting with n nodes and m operations, takes $O(m \log^*(n))$ time instead of $O(m \log n)$ time, where the $\log^*$ function is the iterated logarithm (also called the super logarithm) and is an **extremely** slow growing function.
- $\log^*(n)$ is defined as follows:
  - 0, if $n <= 1$
  - $1 + \log^*(\log n)$ if $n > 1$

SOME PRACTICAL PROBLEMS

- Terrorism, insider trading, financial fraud analysis
  - Are two people connected given millions of “x knows y” statements?
- Vulnerability Assessment
  - Are two computers in a network connected?
- IC Design
  - Are two points shot circuited on this mother board?
- Click Fraud Analysis, Page Ranking
  - Are two web pages connected (indirectly)?

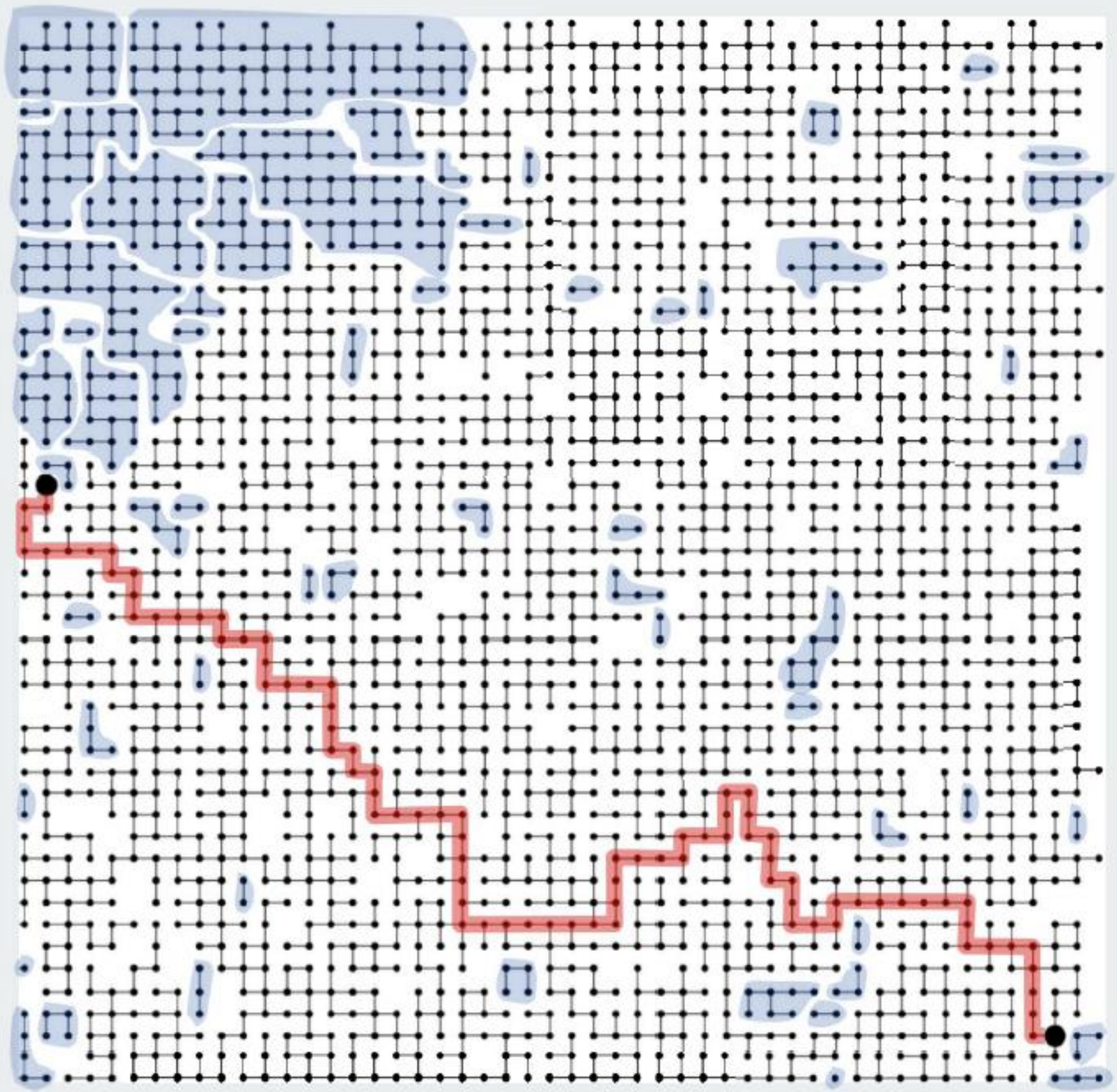
- **Abstractions**
  - **Given a graph, is there a path connecting one node to another?**
  - **How can we organize a given universe of objects into sets?**

`find(u, v) ?`



`find(u, v) ?`

true



READING ASSIGNMENT

- Divide and Conquer
  - http://www.cs.cmu.edu/afs/cs/academic/class/15210-f11/www/lectures/03/lecture03.pdf
  - http://en.wikipedia.org/wiki/Divide_and_conquer_algorithm
- Recursive Algorithm
  http://en.wikipedia.org/wiki/Recursion_(computer_science)
- Tail Recursion
  http://en.wikipedia.org/wiki/Tail_call
- Recurrence Relations
  http://en.wikipedia.org/wiki/Recurrence_relation