CoE 163

Computing Architectures and Algorithms

01a: Algorithms Review

REMEMBER EEE 121?

Data structures and **algorithms** are key in solving any computer engineering problem.

Knowledge of these basic concepts enable you to solve large real-world problems.

BASIC DATA STRUCTURES

- Basic
  - Numbers
  - Strings
  - Sets
- Linear
  - Arrays, linked lists
  - Stacks, queues
- Graph
  - Adjacency matrix
  - Adjacency list
  - Disjoint set

BASIC DATA STRUCTURES

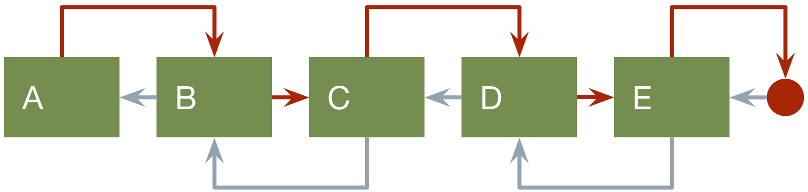
- Trees, heaps
  - Binary tree (AVL, red-black)
  - String trees (trie)
- Geometry
  - Point pairs
  - Polygon list

DATA STRUCTURES: LINEAR

Array: elements of usually same type arranged linearly

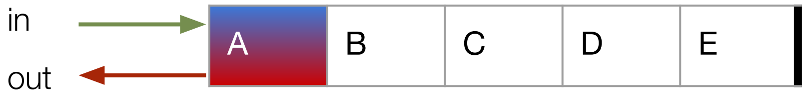
A	B	C	D	E

Linked list: a loosely-connected array



DATA STRUCTURES: LINEAR

Stack: last in, first out; single-ended array



Queue: first in, first out; double-ended array

DATA STRUCTURES: LINEAR

- Arrays are useful for fixed and arranged things
  - AA battery chargers
  - Piano keys
- Linked lists are useful for things where middle elements can change
  - Clinic appointments with cancellations
  - Word editing (letter insertion/deletion)

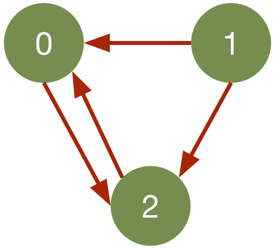
DATA STRUCTURES: LINEAR

- Stacks are useful for things that need stacking - usually vertical
  - Box stacking in warehouses
  - Tetris
- Queues are useful for things that fall in line - usually horizontal
  - Queueing systems in fast food
  - Groceries sorted by expiry date
- A stack and a queue in one is called a **deque** (double-ended queue)

DATA STRUCTURES: GRAPH

Adjacency matrix: 2D array with indices as the two nodes and value the weight or interconnection flag

		destination		
		node 0	node 1	node 2
origin	node 0	0	0	1
	node 1	1	0	1
	node 2	1	0	0

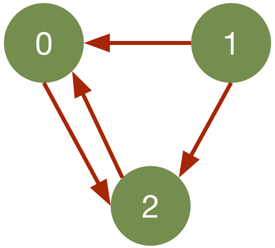


- Rows correspond to origin node and columns the destination node
  - Can be reversed depending on how you code the graph
  - An undirected graph can be represented as a symmetric matrix
- If an element along the diagonal is nonzero, there is an edge to the element itself

DATA STRUCTURES: GRAPH

Adjacency list: Array of variable-length arrays listing neighbors of a node

node	neighbors	
node 0	2	
node 1	0	2
node 2	0	

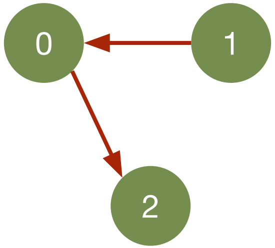


- Saves space as it does not allocate a vxv matrix (v the number of nodes)
- An edge to a node itself can be represented by listing itself in its adjacency list

DATA STRUCTURES: GRAPH

Disjoint set: Array with indices as node labels and value denoting which node is its parent

	node		
	node 0	node 1	node 2
parent	2	0	-1



- This is actually a <u>set data structure</u>, but usually comes up in graphs
- Disjoint sets do not have cycles and have only <u>one parent</u>
- Traversal is recursive and can be implemented efficiently if paths are compressed

DATA STRUCTURES: GRAPH

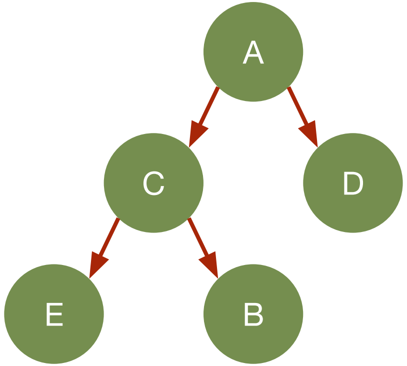
- Adjacency matrices are useful for dense and small graphs
  - “Flow Free” game
- Adjacency lists are useful for sparse and large graphs
  - Road networks
- Disjoint sets are useful for child-parent-like relationships
  - Family trees

DATA STRUCTURES: TREE

Binary tree: Tree that has at most two children

node

0	1	2	3	4
A	C	D	E	B



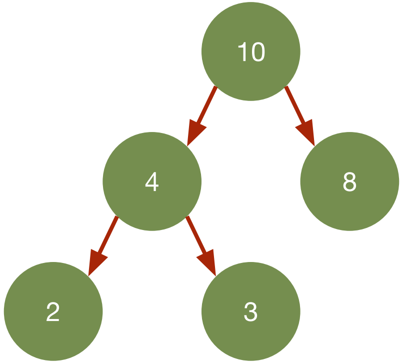
- There are different kinds of binary trees
  - AVL, red-black, splay…
- Balancing is important to ensure efficient traversal and mutation
- Implement as a graph, linked “list”, or 1D array
  - Linked “list” consists of nodes, with each tracking the left and right subtrees
  - 1D array arranged as breadth-first traversal

DATA STRUCTURES: TREE

Heap: Tree that satisfies the heap property (parent root has higher/lower value than children)

node

0	1	2	3	4
10	4	8	2	3



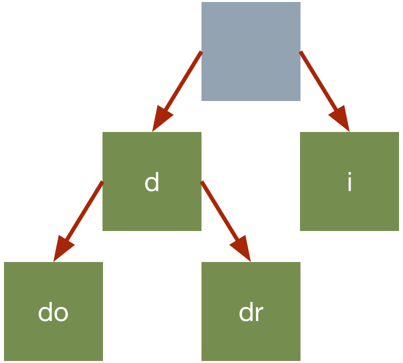
- Max heap has the highest-valued node at the root
- Can be stored as a 1D array the same as a binary tree
- Balancing is important to maintain the heap property

DATA STRUCTURES: TREE

Trie (Prefix tree): Tree that locates specific keys within a set

node

0	1	2	3	4
	d	i	o	r



- A node is defined by its parent prefix and its value concatenated
- Can be stored as a 1D array with the suffix as value
- Children of leaf nodes need to be represented with a symbol to denote end of trie

DATA STRUCTURES: TREE

- Binary trees are useful for balanced matching and searching
  - Parentheses matching
- Heaps are useful for maintaining order while mutating data
  - Senior citizen lane in groceries
- Tries are useful for matching and finding
  - String searching

BASIC ALGORITHMS

- Graph theory
  - Traversal and shortest paths
  - Minimum spanning tree
- Problem solving paradigms
  - Complete search and recursion
  - Divide and conquer
  - Dynamic programming/greedy

BASIC ALGORITHMS

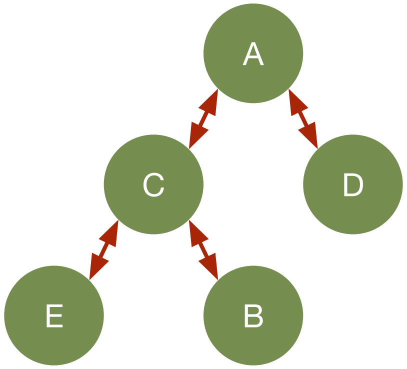
- Math and geometry
  - Probability and statistics
  - Plane/analytic/spherical geometry
- String processing
  - String matching
  - Trees, tries, and arrays
- Data processing
  - Sorting
  - Filter and transformation

ALGORITHMS: GRAPH TRAVERSAL

Graph traversal: Search by visiting a node and its neighbors systematically

Traversal order

- DFS: A-C-E-B-D
- BFS: A-C-D-E-B



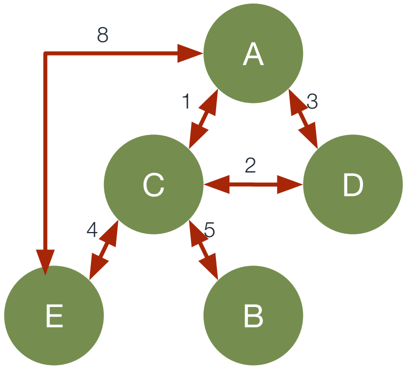
- Depth-first search (DFS): visit the deepest part of a path, then backtrack
  - Uses a stack to track nodes being visited
- Breadth-first search (BFS): visit by layer
  - Uses a queue to track nodes being visited
- Traversal can be modified to determine shortest path between two nodes

ALGORITHMS: SHORTEST PATH

Shortest path: Find path between two nodes that has the minimum weight

Some shortest paths

- A to D: costs 3 (A-D)
- A to E: costs 5 (A-C-E)



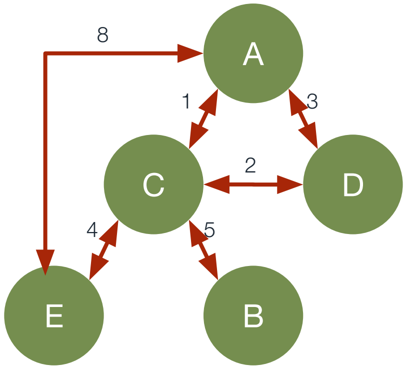
- Dijkstra's: visit and "relax" edges to find minimum-weighted path
    - A priority queue can be used to pick which nodes to visit first
- Bellman-Ford: similar to Dijkstra's but works on negative weights
    - Uses dynamic programming to "relax' edges

ALGORITHMS: MINIMUM SPANNING TREE

Minimum spanning tree: Find set of edges that cumulatively have the total minimum weight and still connects all nodes

Minimum edges needed

- With weights 1, 2, 4, 5



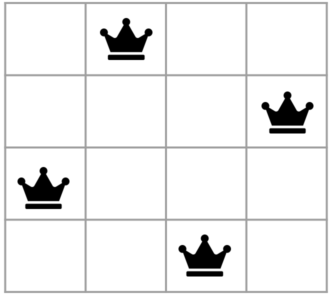
- Kruskal: Sort edges from the lowest weight and get those on top if the two nodes are not connected yet
- Prim: Select a random node and pick a connecting edge with the lowest weight. Collect edges from the resulting connected node and repeat choosing of edges *among all the connected nodes* in such fashion.

ALGORITHMS: PROBLEM SOLVING

Complete search: Iterate through all possibilities of a solution systematically

Put queens on a grid where they do not threaten each other

- Put queens by row, taking care to put them in separate columns
- Check for threats at the diagonal and backtrack to previous layout if there are



- Brute force: Create nested loops or recursions to explore all possibilities
- A*: Explore nodes with the lowest cost first
- Graph traversal: Reform problem into a graph problem and traverse through all possibilities

ALGORITHMS: PROBLEM SOLVING

Divide and conquer: Recurse through a problem by splitting it into n similar problems and consolidating the solutions

You have a cat of length a. Find two cats on a row of cats ordered from shortest length that is a little longer and little shorter than yours.

- Use binary search to find the floor and ceiling lengths

row of cats

1	2	3	5	7	10	12

Your cat is of length 8
Go to segments 5-12, 5-7, 10-12
Closest lengths are 7 and 10

- Binary search on a tree is an example
- Bisection method is useful in arriving at a numerical solution

ALGORITHMS: PROBLEM SOLVING

Dynamic programming: Prune complete search by observing recursion leading to an optimal solution

Determine grouping of matrix chain multiplications that will yield the smallest number of operations

- A: 3x2, B: 2x5, C: 5x4; E = ABC
- Group matrices like complete search
- Overwrite saved solution if it is smaller

from \ until	A	B	C
A	0	30	64
B		0	40
C			0

- <u>Top-down</u>: Recurse from the top and parts of the solution for later rebuilds
- <u>Bottom-up</u>: Build up to the solution from base cases
- <u>Build order is important!</u>
- Solution configuration can be recovered by saving previous iterations

ALGORITHMS: PROBLEM SOLVING

Greedy: Get what is best at the moment

Buy two take one free promo! Find the maximum discount you can get given your basket items.

- Go to the counter with the three most expensive items on your basket every time

Saved 9!

- Special case of dynamic programming that satisfies the greedy property
- Although it does not work all the time, it can yield fast and slightly suboptimal solutions
- When in doubt, use dynamic programming instead

ALGORITHMS: MATH AND GEOMETRY

Basic arithmetic: Remember elementary axioms, factors, etc.

Three friends share a garden - one worked A hours and another worked B hours to clean up the whole garden. The third friend paid D dollars. How much should A get?

$$S_A = \left(A - \frac{A+B}{3}\right)\frac{D}{\frac{A+B}{3}}$$

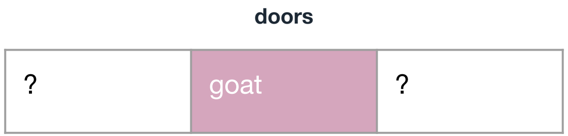
A, B, and C have equal shares. A and B clean up their respective areas plus extra time that they give up to clean C's area.

- Radix/Base conversion
- Numerical pattern finding
- Fractions
- Logarithms and exponents
- Prime numbers
- Modular arithmetic
- Euclidean algorithm

ALGORITHMS: MATH AND GEOMETRY

Probability and statistics: Apply basic probability axioms and combinatorics

Monty Hall problem - find the chance of winning when you switch to another door



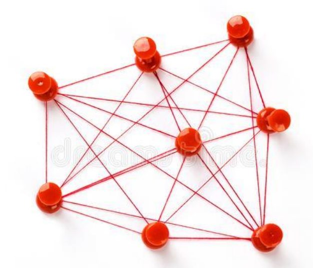
If you stayed with your original choice, it's as if you just opened that door straight away, so the chance is ⅓.

The chance of switching is therefore ⅔.

- Permutations and combinations
- Bayes' theorem, conditional probabilities
- Binomial, Catalan, Fibonacci numbers

ALGORITHMS: MATH AND GEOMETRY

Geometry: Apply 2D and 3D geometry theorems and conjectures



- Line and plane intersections
- Area, perimeter, volume
- Convex hull
- Point inside polygon
- Be careful of <u>numerical errors</u> when using floating-point!

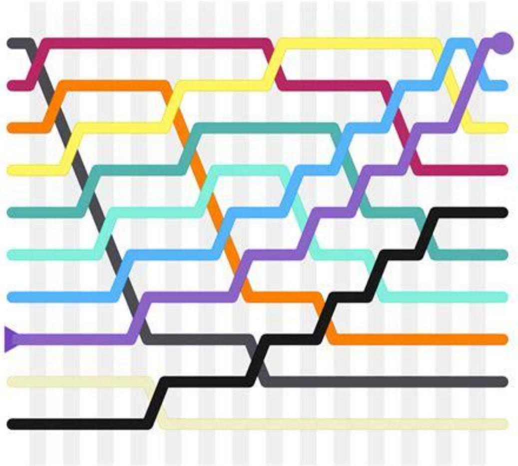
ALGORITHMS: STRINGS

String matching, processing, and manipulation

- Knuth-Morris-Pratt algorithm
- String alignment
- Suffix trie, prefix tree, arrays

ALGORITHMS: DATA PROCESSING

Apply algorithms to sort, filter, and transform data



- Bubble, insertion, selection sorts
- Priority queue
- Summation and production
- Bit masking
- Character to ASCII value

TIPS

- Learn new algorithms and data structures
- Be exposed to a lot of known CS problems
- Practice by trying out online judges, solving some problems, and getting used to input/output formatting

CoE 163

Computing Architectures and Algorithms

01a: Algorithms Review