COMP 250: Intro to Comp Sci

Julian Lore

Last updated: April 16, 2017

Contents

1 Interfaces 03/14/17

Java interface is similar to a class, but only has method signitures. Does not actually implement anything, just lists methods and what they return. Can implement with a class. Cannot be instantiated. Can have a generic interface, with <T> where T is the type of object stored.

Implementing an interface Class implements interface by providing code for each method of the interface. Can have extra methods too. Can also have generic class with <T>.
class ArrayList<T> implements List{}
Can instantiate by: ArrayList<String> myList = new ArrayList<String>();

The Java Collections Lots of interfaces in Java, with hierarchies, some interfaces exttending other more generic ones and then some classes implementing these.

Iterator interface Used to traverse a collection of objects, hasNext and next method.

Comparable interface Can compare with inequalities/equality.

2 Graphs 03/16/17

Will be talking about graphs for the next 3 or 4 lectures. Kind of like a generalization of the data structures we've seen.

Graphs Pair (V, E) with V being the set of nodes called **vertices** and E is a collection of pairs of verties, called **edges**.

Edge Types

- **Directed edge** ordered pair of vertices (u, v). First vertex u is origin, second, v is destination.
- **Undirected edge** unordered pair of vertices (u, v).
- **Directed graph** all edges directed.
- **Weighted edge** has real number associated to it (like distance).
- **Weighted graph** all edges have weights.

Labeled graphs Vertices have identifiers (names), geometric layout doesn't matter, only connections.

Unlabelled graph Vertices have no identifiers.

Terminology

- **Endpoints of an edge** The 2 vertices at the end of an edge.
- **Edges incident on a vertex** Edges that have that vertex as an endpoint.
- **Adjacent vertices** Connected by an edge.
- **Degree of a vertex** Number of incident edges.
- **Parallel edges** Edges that have the same endpoints, multi edge graph that counts both for degree.
- **Self-loop** Edge from a vertex to itself.
- **Path** Sequence of adjacent vertices.

- **Simple path** Path such that all vertices distinct
- Graph is **connected** $\iff$ $\forall$ pairs of vertices u and v, $\exists$ path between u and v. If there are directions, have to take that into account, might be able to go one way but not another way.
- **Cycle** path that starts and ends at same vertex.
- **Simple cycle** Cycle where each vertex is distinct.
- A tree is a **connected acyclic** graph.

Degree

1. $$\sum_{v} \in vdeg(v) = 2|E|$$

   (for undirected graphs) each edge contributes to degree of 2 nodes.

2. In an undirected graph with no self-loops and no multiple edges

   $$|E| \leq |V|(|V| - 1)/2$$

Data structures for graphs Graph can be stored as

- Dictionary of pairs (key,info) with key=vertex identifier, info contains list called adj of adjacent vertices

Implementation as linked-list is redundant, accounts for each edge twice, as each vertex specifies which vertices it is adjacent to. For searching, need to search for the vertex in linked list and then search through contents of vertex.
Implementation as adjacency matrix, good at everything linked list implementation is bad at and bad at what it was good at.
Graph with n vertices is stored as

- $n \times n$ array M of boolean with
- $M[i][j] = \begin{cases} 1 & \text{if there is an edge between ith and jth vertices} \\ 0 & \text{otherwise} \end{cases}$
- Still redundant, matrix is symmetrical, unless directed.

- Diagonal is arbitrary, if we put a 1 it would imply vertices have a self-loop, but we assume nodes are incident to themselves.
- Not good for graph with parallel edges, but good for weighted graphs, can change 1 to weight.
- Another drawback is that matrix can become very large. A million by a million, too large to fit. Would have to use adjacency list as those only use memory for vertices that are incident to each other.

3 Review April 7th, 2017

- Final Exam April 18 at 2 pm
- Multiple choice
- Covers entire semester
- Double-sided crib sheet
- No calculators
- 50%
- 29 questions
- Blanchette won't be there during the exam
- If there is a problematic question, answer it as best as you can. If it's invalid, will be cancelled later.
- Slight emphasis on 2nd half of the semester, but barely

Going over the first half of the semester today.

- Material for prep
- Lecture notes
- Assignment solutions
- Previous exams
- There will be Java questions like on midterm

- What happens when I call something
- Will be asked to "write algorithms" in Java through multiple choice
- Closely related to assignments

Iterative sorting algorithms

- Selection sort
- Insertion Sort
- Bubble Sort
- Their running times and why

Recursion

- Designing recursive algorithms
  - Breaking problems into smaller sub problems
  - Solve sub problems recursively
  - Combine solutions
  - Base case!
- Dividing original problem into roughly equal size subproblems for (usually) better running times
  - Power
  - mergeSort
  - quickSort (good pivot vs bad pivot)
  - integer multiplication
- Tracing recursion (like hw4)
- Trees: recursion usually easier
- Analysis of run time of recursive algorithms
  - Write recurrences for $T(n)$
  - Solve recurrences with substitution

  - Prove using induction

- What gets printed from a recursive statement?
- Using a stack to model recursion
- Coming up with recursive formula given algo

Proofs by induction & loop invariants

- Proofs by induction
  - Logic
  - Base case
  - Inductive step
- Loop invariants, prove correctness of program
  - Property that holds at every loop
  - Use property to show correctness

Running time & big-Oh

- Running time
  - Counting primitive operations
  - Dealing with loops, summation
  - Worst vs average vs best case
- Big-Oh
  - Mathematical definition
  - Proving relationships
    - Using definition
    - Simplification rules
    - Limit of ratio
    - Big-Oh hierarchy
  - Relevant only for large inputs
    - Can be irrelevant for small (integer mult)

- Big-Theta(what we usually mean when we say big-Oh of an algo), Big-Omega
- Unless mentioned otherwise, big-Oh running time is for worst-case
- Know and understand big-Oh and running time of all algos seen in class

Data structures

- Array
  - Run times for ops
- Single-linked list
  - Can assume they know their size of exam
  - Better than arrays:
    - Insertion, deletion
    - Don't need to know size beforehand
  - Worse than arrays
    - Finding n-th element (binarySearch is hard)
    - More memory (for "next" member)
- Doubly-linked list
  - Can move backward
  - Can delete easier
- Stacks and Queues
  - Understand applications we saw
  - How do they work

4 Review April 11th, 2017

QuickSort

- How does partition work?
- Worst case $O(n^2)$, when? Already sorted
- Average case: $O(n \log n)$
- Randomized choice of pivot average case: $O(n \log n)$

Tree

- treeNode representation
- node, leaf, root, parent, sibling, descendants, ancestors, subtree rooted at x, internal and external nodes, ordered, binary, proper binary
- depth(distance from root), height (distance to furthest leaf, height of tree is height of root), computations
- Tree traversal, pre-order, in-order, post-order

Dictionary ADT

- Stores pairs, keys with info
- Operations: find(key), insert(key,info), remove(key)
- Cases where array implementation is bad
- Cases where linked-list implementation is bad

Binary Search Tree (Dictionary)

- Keys in left subtree have keys smaller than or equal
- Keys in right subtree have keys greater or equal to
- Algorithm to find key is $O(h) = O(\log n)$ if tree is balanced
- Inserting a new key, running time $O(h)$.
- Removing a key
- Might need to execute algorithms by hand

Hash tables (Dictionary)

- Map keys to buckets
- Each bucket is a dictionary
- Hash functions: minimize collisions and easy to compute
- Best case, keys distributed uniformly
- Worst case, all keys end up in same bucket

Priority Queues, Heap

- key(x) smaller or equal to keys of children of x
- All levels except last are full, last level nodes are packed to left
- findMin(0 $O(1)$
- insert(key) bubbling up, $O(\log n)$
- removeMin() bubbling down $O(\log n)$
- HeapSort, insert keys one by one, removeMin() one by one

Java Collections

- Interface
  - List methods to be provided, doesn't implement them
  - Java way to describe ADT
  - Important interfaces, iterator, comparable (has compare method)
  - Implementing interface
    - class ArrayList¡T¿ implements List
  - Using existing collections

Binary

- Conversion to and from decimal
- Number of bits required to store an integer $\lfloor \log_2 N + 1 \rfloor$

Greedy and Dynamic Progrmaming Algorithms

- Greedy, always make choice that seems best in short term. Most of the time non-optimal. For some problems, optimal and fast.
- Dynamic, bottom-up version of a recursive algorithm. Start by oslving small problems, saves results to solve larger. Longest increasing subsequence problem, like making change.

Graphs

- Terminology
- Data structures
  - Adjacency-list
  - Adjacency-matrix
  - Running time of operations
- Graph traversal
  - Depth-first search, recursive or iterative using a stack
  - Breadth-first search, iterative using a queue
- *Important: applications of DFS and BFS
- Search engine algorithm
- Graph problems
  - Shortest path
  - Cycles
  - Graph coloring
  - Cliques and independent-sets
  - Matching

Game strategy

- Single-player
  - Backtracking
  - 8-queens, add one by one and backtrack when invalid position is reached
- Multi-player
  - Game tree
  - Winning and losing positions
  - Minimax principle

Cryptography

- Secret-key schemes and problems
- RSA public-key cryptography
- Bob keeps private key to self
- Alice gets Bob's public key, uses it to encode
- No one knows how to decode in polynomial time without private key

Heuristics

- Useful when don't have fast exact algos
- Greedy
- Fastest descent with randomization, genetic
- Applications to Traveling Salesperson

Ray-tracing algorithm

- Problem addressed by ray-tracing
  - Computer graphics
  - Compute 2D images from 3D objects
- Ray tracing algorithm and recursive ray-tracing
- Quad-tree data structure
  - Subdivide space in quadrants
  - Keep subdividing until regions have at most one object
  - Allows to quickly indentify intersection of ray and objects