Graphs

11-6-2003

Opening Discussion

- What did we talk about last class?
- Do you have any questions about the assignment?
- What is a linked list? What are limitations on it? What is a tree? What are the limitations on them? Can you describe a linked structure (or draw a picture) which is neither a tree, nor a list?
- AVL vs. Red-Black, Quad vs. KD

B-Trees and Disk

- The book covers this a bit, but we should discuss it as well. In a B-tree the way we reference children is with file locations.
- So a node has one vector of keys (and maybe data), and one vector of "links" which are locations in the file. The links vector is always one longer than the keys vector.
- Whenever we are working with a node, we jump to the location in file and read it. If we make changes, we jump to the same location and write.
- Never keep many nodes in memory.

What is a Graph?

- A graph, G(V, E), is made from two sets.
- The first set, V, is a set of vertices (I might call them nodes occasionally).
- The second set, E, is a set of edges which are ordered pairs (u,v) such that u and v are elements of V.
- Basically, anything you can draw on a board with dots/boxes and lines is a graph.

Variations on a Theme

- There are many terms used to describe different "types" of graphs.
  - Sparse vs. dense - if there are close to V^2 edges a graph is dense.
  - Di-graph - short for directed graph. In a non-directed graph if E contains (u,v) it also contains (v,u).
  - Cyclic vs. Acyclic - is there a path from u back to u without repeating an edge?
  - Connected vs. disjoint
  - DAG = Directed Acyclic Graph
  - Weighted graphs - edges have a value associated with them.

Adjacency Lists

- One way to store a graph is with a set of lists. We have one list for each vertex and that list contains all the vertices it has edges to.
- This is good for sparse graphs because we only use memory proportional to the number of edges.
- The downfall is that you can't quickly ask if a certain pair of vertices is joined by an edge.
- Storing edges in nodes is basically equivalent to this.

Adjacency Matrix

- An alternate method of storing a graph is with an adjacency matrix. Here, every vertex is given a number. If there is an edge (u,v) then the element at row u, column v has a 1 in it, otherwise it is 0.
  - For a weighted graph the value is the weight, not just 1.
- This gives fast lookup of edges, but requires $O(V^2)$ storage even for sparse graphs.

Breadth-First Searching

- This is a traversal of a graph where we discover things at the same level in order. To prevent infinite loops in cyclic graphs we color nodes. The book uses three colors to help with some proofs and algorithms, only two are really needed.
- The breadth-first search uses a queue to store nodes that have been discovered. All nodes start white, are made gray on discovery and black when all children have been enqueued.
- Can store depth and parent information.

Depth-First Searching

- This style of search uses recursion (a stack). To help with some extra algorithms, the book has this store discovery time and finish time for each node. Again a three color scheme is used to prevent infinite loops and with proving of certain aspects of the algorithms.

Topology Sorting

- For a DAG, a depth first search can easily produce an ordering for all the vertices such that if there is an edge (u,v) then u comes before v in the ordering.
- This ordering can be made by sorting the elements in reverse order by their finishing time. This is done quickly by simply inserting the nodes at the beginning of a linked list when finished.

Strongly Connected Components

- A strongly component of a graph is a maximal subset, C, of V such that for every pair of nodes, u and v, in C there is a path from u to v and a path from v to u.
- To find these we do a DFS of G, then computer G^T where all the edges have been reversed. Now we do a DFS of G^T, but visit nodes in order of descending finishing time. The trees in the forest of that traversal are the strongly connected components.

Minute Essay

- What did we talk about today? You should turn in your test code for assignment #4 today (or soon) and remember that the whole thing is due on Tuesday.
- I'll be out of town and likely without much e-mail for the weekend due to the ACM programming competition.