Greedy Algorithms

2-7-2006

Opening Discussion

- Do you have any questions about the assignment?
- Do you have any questions about the reading?

Searching Solution Space: Issues

- We already talked about how one could approach the solving of many different problems with algorithms that search through the full solution space looking for the ideal solution. Often this is done with recursive algorithms.
- The primary problem with this is that solution space could be quite large and that might take an intractable amount of time.
- For some problems we can do things more efficiently. The real trick is figuring out what the most efficient method is for a particular problems.

Greedy Algorithms

- Greedy algorithms are some of the least general algorithms for solving optimization problems. However, when they can be used they are extremely fast, often taking only O(n) time.
- The general idea of greedy algorithms is that at every step they pick the option that is most ideal and they never look back to reconsider that decision. This is what makes them both so fast and so inflexible.

Event Scheduling

- Also called interval scheduling, the problem starts with an input of a number of intervals of time that you need to schedule. You can't schedule overlapping events/intervals.
- In this case our objective is to schedule as many of the events as possible. What algorithm might we use for doing this? What would it look like to search the solution space? What order would that be?

Greedy Solution

- We can solve this problem in a greedy way by selecting events in order of the time they finish without taking overlapping events. First sort the list by finishing time and pick the first element, then proceed down the list taking the first event you come to that doesn't overlap the previous one.
- This is O(n log n) because of the sort at the beginning. The processing after the sort is O(n).
- You can show that this is optimal by noting that any optimal solution can't have more events taking up less total time starting at the beginning.

Interval Partitioning

- This is a very similar problem to the event scheduling. Only now we have multiple resources and we are going to schedule everything. The question here is, what is the fewest number of resources we can use.
- This problem is also called interval coloring. We want to know the minimal number of colors we have to use so that no overlapping intervals have the same color.
- This problem can be solved with an $O(n^2)$ greedy algorithm. We sort the intervals by starting time and run through each one. At each one we find a value that hasn't been used in and overlapping partition and give it the minimum such value.

Scheduling on Deadlines

- A slightly different problem is one where we go back to having a single resource, but now our tasks can float. We know how long each task takes and we have a deadline for each task. Our objective is to minimize the maximum lateness while completing all tasks. We could pick other goals, but they wouldn't have greedy solutions.
- The solution is simply to order the tasks by their deadline time and do them in that order. This simple solution produces an ideal result, but proving it does so is a more complex issue.

Cache Maintenance

- The idea for this problem is that you have a cache, like on a computer, and you know exactly the sequence that items will be requested. You want to figure out what you place back in cache each time there is a miss to minimize the total number of misses.
- Given this situation, it turns out that an optimal solution can be found by always evicting the items that will be needed at the furthest time in the future.

Huffman Codes

- One of the most common examples of a greedy algorithm is that of Huffman codes. The idea is that you want to encode a message in a more compact form than standard encoding.
- For example, normal text in a machine uses 8 bit chars, but we don't really use 256 options, end even if we did, some things, like 'e', occur so much more than others that it would be better to use fewer bits for their representation.
- Huffman codes are optimal variable bit codes for messages. Each symbol has a set of bits of variable length. Common symbols require fewer bits. It is optimal in that the full message takes the fewest bits possible.

Generating Huffman Codes

- So given a message, how does one produce the Huffman code for it? First, count the frequency of each symbol and make a set of “trees” where each symbol is a single node.
- Repeatedly you pick the two roots whose probability is lower and merge them into a single tree. When you get to one tree you have a Huffman code. It is a binary tree where going left is a 0 and going right is a 1.

Greedy Approximations

- Some problems can't be solved optimally with greedy algorithms, but we can use greedy algorithms to get approximate solutions and we can often prove bounds on how close that will be to the optimal solution.
- TSP

Reminders

- Remember to turn in assignment #2 today at some point.