Dynamic Programming

2-21-2006

Opening Discussion

- What did we talk about last class?
- Do you have any questions about the assignment?
- As a hint on this and other assignments, you should write methods that validate the internal structure of the data structure and call them liberally when you have bugs.

When to Memoize?

- In general we want to memoize any time when we see that our recursive solution is calculating the same thing more than once. This is common when a problem exhibits optimal substructure.
- As we will see next time, we can use dynamic programming instead of memoization when we have optimal substructure. DP will be slightly more efficient but memoization might be more intuitive.
- Memoization can also be used in situations where DP can't because subproblems aren't solved in a nice order or because we don't have true optimal substructure.

Memoizing TSP

- This latter case is exemplified by the traveling salesman problem. It does not have optimal substructure. However, the standard recursive solution to this does repeat work.
- It is tempting to do this with “smart breadcrumbs”, but that isn't correct. It could return non-optimal results. We need to use memoization so we store the length of completing the rest of the cycle from a given state?
- What is our state and how does that index into an array?

Dynamic Programming

- When a problem exhibits optimal substructure it is possible to convert the recursive algorithm to an iterative one that will solve the problem in provably polynomial time.
- Once again, optimal substructure is the property of a problem where the optimal solution to a full problem is composed of optimal solutions to smaller problems.
- The idea with dynamic programming is that we will fill in a table of solutions from the bottom up so that at each new element we are always using values we have already calculated.

Constructing a DP Solution

- The method to create a DP solution to a problem begins the same way as searching solution space or memoization, with the description of the problem in terms of a recurrance relationship. The trick is to find the right recurrance.
- Once we have that we make a table with as many dimensions we we have arguments to the recursive function and deal with the simple boundaries. Then we loop through the table filling in values by looking things up in the table.
- The last step, if needed, is to reconstruct the solution by working back down the table from top to bottom.

Weighted Interval Scheduling: DP

- Let's revisit the problem that we did last time of scheduling weighted intervals to maximize the weights of the events that are scheduled.
- First we want to write down the recursive function.
- Once we have that we can use an array to store values and build them up from the bottom.

Longest Common Subsequence

- This is probably the most common example of dynamic programming. Given two strings, you want to find the longest subsequence that they have in common. A subsequence doesn't have to be consecutive characters, it just requires that the characters appear in the proper order.
- Let's build a DP solution for this problem.

0/1 Knapsack

- This is another classic DP problem. Again we will begin by writing the recursive function.
- What happens to us if weight isn't an integer? How must we change our code to make it work?

Reminders

- Next time we will look at some other DP problems that you likely haven't seen before.
- Remember to turn in assignment #3 today.