Introduction to Dynamic Programming

Different Problem Solving Approaches

Greedy Algorithms

- Build up solutions in small steps
- Make local decisions
- Previous decisions are never reconsidered

Dynamic Programming

- Solves larger problem by relating it to overlapping subproblems and then solves the subproblems
  - Important to store the results from subproblems so that they aren't computed repeatedly
  - (overlapping: contrast with divide and conquer, which divides into independent subproblems)

Backtracking

- Solve by brute force searching the solution space, pruning when possible

Dynamic Programming

General Idea:

- Solves larger problem by relating it to overlapping subproblems and then solves the subproblems
- It works through the exponential set of solutions, but doesn't examine them all explicitly
- Stores intermediate results so that they aren't recomputed

Dynamic Programming

For dynamic programming to be applicable:

- At most polynomial number of subproblems (else still exponential-time solution)
- Solution to original problem is easily computed from the solutions to the subproblems
- There is a natural ordering on subproblems from "smallest" to "largest" and an easy to compute recurrence that allows solving a subproblem from smaller subproblems

Dynamic Programming – A First Example

Fibonacci Numbers

- 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …
- F(0) = 0, F(1) = 1
- F(n) = F(n-1) + F(n-2)

Computing the Fibonacci Numbers

- Each n^{th} number is a function of previous solutions
- A recursive solution:

```
Fib(n)
1. if n < 0 then RETURN “undefined”
2. if n ≤ 1 then RETURN n
3. RETURN Fib(n-1) + Fib(n-2)
```

What’s the drawback to this solution?

- Complexity is exponential

Dynamic Programming – A First Example

Computing Fibonacci Numbers – Can we do better than exponential?

- Yes – "Memoization"
- Each time you encounter a new subproblem and compute the result, store it so that you never need to recompute that subproblem

- Each subproblem is computed just once, and is based on the results of smaller subproblems
  - This leads naturally to converting the recursive solution to an iterative solution

```
FibDynProg(n)
1. Fib[0] = 0
2. Fib[1] = 1
3. for i=2 to n do
4.    Fib[i] = Fib[i-1] + Fib[i-2]
5. RETURN Fib[n]
```