CSE 101 Class Notes

A simple example of dynamic programming

November 20, 2006

Dynamic Programming

Fibonacci numbers are defined recursively as

$$\begin{aligned} F(0) &= F(1) = 1 \\ F(n) &= F(n-1) + F(n-2) \end{aligned}$$

You can turn this into a recursive implementation which runs in time $T(n) = T(n-1) + T(n-2) = F(n)$ (which is exponential). Notice that this algorithm recomputes a lot of intermediate results (e.g. $F(n) = F(n-1) + F(n-2) = F(n-2) + F(n-3) + F(n-2)$). This suggests that we store the intermediate values $F(i)$ and re-use them. Following this intuition, we can define an efficient recursive ("memoized") implementation:

```
Fib <- { array 1..n of results }
fib(n)
    if n < 2
        return 1
    else
        fib(n - 1)
        Fib[n] <- Fib[n-1] + Fib[n-2]
        return Fib[n]
```

Dynamic programming combines storing intermediate results with a second idea: instead of computing the subproblems top-down, we compute them bottom-up:

```
fib(n)
    Fib <- { array 1..n of results }
    for i = 2 .. n
        Fib[i] <- Fib[i-1] + Fib[i-2]
    return Fib[n]
```