Dynamic Programming

Rajat Khanna

April 2015

1 Introduction

Dynamic Programming, hereinafter DP, is a a technique used to optimize algorithms, generally reducing time from exponential to polynomial, at the expense of memory. The basic idea behind DP is that if a problem can be divided into smaller sub-problems with the same solution, then DP can be used to prevent recomputation and speed up the overall solution. For those of you who already know DP (which will be most), I encourage you to skip to section 6 and solve those problems.
Credit for this lecture goes to Ryan Jian, former SCT officer, as his lecture is taught me DP.

2 Example

One of the basic examples of DP can be seen for computing the fibonacci sequence.

```
def fib(N) {
    if (N == 0 || N ==1)
        return 0;
    return fib(N-1) + fib(N-2);
}
```

We see that the recursion here calls itself twice, giving us an exponential runtime. We also note that certain computations are repeated multiple times for larger values of N.
To improve upon this, we introduce a *cache* array to store previously computed values, allowing for a significant speed up.

```
cache[] = [N+1]
def fib(N) {
    if (N == 0 || N == 1)
        return N;
    if (cache[N] != -1)
        return cache[N];
```

```
    cache[N] = fib(N-1) + fib(N-2);
    return cache[N];
}
```

The next thing to note is that this can be further optimized using the bottom-up approach. Rather than starting with the final value and then subdividing, we can start with the smallest problem and work up to the desired value.

```
def fib(N) {
    cache[] = [N+1];
    cache[0] = 1;
    cache[1] = 1;
    for (i = 2; i <= N; i++_
        cache[i] = cache[i+1] + cache[i+2];
    return cache[N];
}
```

3 The Quintessential DP Problem Everyone Should Know and Cherish

We now look at the most common and recurring DP problem known as the **Knapsack Problem**.
Given N objects, each with some weight $W[i]$ and value $V[i]$, compute the maximal value of a total weight C.// We need to see if we can divide this into a series of repeatable subproblems. Usually, this is the hardest step of the entire process.
Notice that adding a rock is analagous to reducing the capacity of the backpack. That means that every time, we want to add an object such that the sum of the value of the object and the value of the smaller capacity backpack to be maximal. Thus, we have a repeating a problem. We can now determine a recursive relation: $cache[C] = max(V[i] + cache[C - W[i]])$.

```
cache[] = [N+1];
def knapsack(W[N], V[N], C) {
    if (cache[C] != 0)
        return cache[C];
    for (i = 0; i < N; i++)
        if (C-W[i] > 0)
            cache[C] = max(cache[C], V[i] + knapsack(W, V, C-W[i]));
    return cache[C];
}
```

Challenge: Write the bottom-up solution to this problem.

4 Solving DP problems

Ryan outlined 3 major steps to solve any DP problem:

1. Create your state variables: This is mostly a result of intuition and practice. For instance, for the knapsack problem, we see that the only state variable is weight, thus we have 1 state variable. The more variables, the larger the runtime, meaning we want as few state variables as possible.
2. Base Case: The base case is generally given by terms of the problem. For instance, for the knapsack problem, the backpack cannot have a weight less than 0. In terms of recursion, we have that the 0th term is 0 and the 1st term is 1.
3. Recursive relation: This will relate the solution of the larger problems to the smaller subproblems.

5 Optimizations

Several optimizations for DP problems exist as well. I don't know the specifics for them, so you should look them up. Here's a list of what I know exists:

1. Convex Hull Optimization
2. Divide and Conquer
3. Knuth Optimization

6 Practice Problems

There are some well-known problems regarding DP, such as the knapsack problem available at the online list compiled by Brian Dean at: (http://people.csail.mit.edu/bdean/6.046/dp/). Here are some more examples just to practice with.

1. A split operation is defined as transforming a word $w = xy$ into a word $u = yx$. For example "wordcut" can become "cutword". Given two words, start and end, both of length N, ($2 \leq \mathrm{N} \leq 103$), count how many ways start can be transformed into end using K, ($0 \leq \mathrm{K} \leq 105$) split operations. (Codeforces Croc Champ 2012 - Round 2, Word Cut)
2. Given a series of light switches, define a flip operation $f(i, j)$ to flip all the switches from i to j, inclusive. Find the maximal number of "on" switches after exactly 1 flip operation. (Note: Define the light switches as having states 0 or 1, designating 1 to be the "on" state and 0 to be the "off" state). (Codeforces 327A)
3. You've got an $n \times m$ pixel picture. Each pixel can be white or black. Your task is to change the colors of as few pixels as possible to obtain a barcode picture. A picture is a barcode if: All pixels in each column are of the same color, and the width of each monochrome vertical line is at least x and at most y pixels. (Codeforces 225C)