CS 341 — Lecture 9

Bartosz Antczak *Instructor: Bin Ma* *October 5, 2017*

9.1 More on Dynamic Programming

Dynamic Programming involves recursing on a list of subproblems (or sub-instances). It works like recursion except it has a bottom-up approach compared to recursion's top-down approach. Today we'll look at more algorithms using dynamic programming.

9.1.1 Subset Sum

- **Input:** n positive integers $a_1, \ldots, a_n$ and a value $K > 0$
- **Output:** subset S such that $\sum_{i \in S} a_i = K$

For instance, given $[1, 3, 4, 5]$ and a value of 7, a possible subset to return is $[3, 4]$.
Let's define $D(i, k)$:

$$D(i,k) = \begin{cases} \text{true} & \text{if } a_1, \ldots, a_n \text{ has subset } S \text{ such that } \sum_{i \in S} = k \\ \text{false} & \text{otherwise} \end{cases}$$

Let's build a subproblem of size $i - 1$ and consider a_i.

- **CASE 1:** *a_i is used.*
  Then we have $D(i) = D(i - 1, k - a_i)$
- **CASE 2:** *a_i is not used.*
  Then we have $D(i) = D(i - 1, k)$

Our recurrence relation then becomes

$$D(i,k) = D(i-1,k) \text{ OR } D(i-1,k-a_i)$$

Our algorithm is then constructed as such

---

for $i = 1$ **to** n **do**
  for $k = 1$ **to** K **do**
    $D(i,k) = D(i-1,k)$ OR $D(i-1,k-a_i)$

9.1.2 Backpack Problem

- **Input:** n items, each with with weight w_i, value v_i, and a backpack of weight capacity W.
- **Output:** a subset S such that $\sum_{i \in S} w_i \leq W$ and maximize $\sum_{i \in S} v_i$

Notice that the greedy algorithm will not work here, so we must use another technique, perhaps *dynamic programming*?
Let $D(i, w)$ be the maximum value one can achieve with the first i items and capacity w. In the optimal solution, two cases arise for $D(i, w)$:

- **CASE 1:** *item i is used.*
  In this case, $D(i, w) = v_i + D(i-1, w-w_i)$
- **CASE 2:** *item i is not used.*
  Here, $D(i, w) = D(i-1, w)$

We then outline our recurrence relation as such:

$$D(i, w) = \max \begin{cases} D(i-1, w) \\ D(i-1, w-w_i) + v_i \end{cases}$$

Running our trivial dynamic programming algorithm on this recurrence relation yields a runtime of $O(n \cdot W)$ (which is a pseudo-polynomial!)

9.1.3 Longest Common Subsequence

- **Input:** two strings S and T
- **Output:** longest common sequence

For instance, $S =$"utoronto" and $T =$"uwaterloo". The longest common subsequence is "utroo" (recall that a subsequence is *not* a substring — we can skip over letters). A common usage of this kind of problem is to detect plagiarism and commonalities between two papers or code submissions.
Let's break this problem down and focus on the prefixes of both S and T. Define $D(s, t)$ as the length of the longest common subsequence (LCS) of $S[1, \ldots, s]$ and $T[1, \ldots, t]$. Three cases arise:

- **CASE 1:** *$S[s] = T[t]$ and both characters are used.*
  Here, we have $D(s, t) = 1 + D(s-1, t-1)$
- **CASE 2:** *$S[s]$ is not used.*
  In this case, $D(s, t) = D(s-1, t)$
- **CASE 3:** *$T[t]$ is not used.*
  In this case, $D(s, t) = D(s, t-1)$

And so our recurrence for $D(s, t)$ is

$$D(s, t) = \max \begin{cases} 1 + D(s-1, t-1) \\ D(s-1, t) \\ D(s, t-1) \end{cases}$$

Our algorithm to compute it will have a run time of $O(nm)$ (n is length of S, and m is length of T).

9.1.4 Edit Distance

We define a set of string operations:

- Add a letter at index i
- Change a current letter at index i
- Remove a letter at index i

Our problem is outlined as:

- Two strings T and S
- The minimum number of edit operations required that convert S to T

Define $D(s,t)$ as the edit distance between $S[1,\ldots,s]$ and $T[1,\ldots,t]$. Here, four cases arise:

- **CASE 1:** $S[s] = T[t]$.
  In this case, $D(s,t) = D(s-1,t-1)$
- **CASE 2:** $S[s] \neq T[t]$ *and* $T[t]$ *is substituted.*
  In this case, $D(s,t) = D(s-1,t-1)+1$
- **CASE 3:** $S[s]$ *is deleted.*
  In this case, $D(s,t) = D(s-1,t)+1$
- **CASE 4:** $T[t]$ *is inserted.*
  In this case, $D(s,t) = D(s,t-1)+1$

$$D(s,t) = \min \begin{cases} D(s-1,t-1) \\ D(s-1,t-1)+1 \\ D(s,t-1)+1 \\ D(s-1,t)+1 \end{cases}$$

From here, we simply construct our algorithm and observe that it has a run time of $O(mn)$ ($m = |S|$, $n = |T|$) .