10

June 16, 2016

1 Dynamic Programming

- form subproblems
- get recursive formula to explore all choices at each step
- evaluate formula bottom-up using a table works when total # subproblems is not too big

Ex 1: Evaluate $C(n, k) = C(n-1, k-1) + C(n-1, k)$ (or 1 iff $k = 0$ or $k = n$) that is the binomial theorem.

Naive is $O(2^n)$. Can optimize to $O(n^2)$ if we cache values since there are at most $O(n^2)$ (n, k) combos.

1.1 Coin Changing

Given coin values c_i, target W, find minimum # of coins that sum exactly to W.

1.1.1 First Solution

Define $C(i, j)$ be min # of coins from $\{c_1 ... c_i\}$ that sum to j. Note the set is ordered.

Then $C(i, j) = \min\{C(i-1, j), C(i, j - c_i) + 1\}$ (what'd happen if the last coin we took is c_i).

Of course, we need to test the bounds to ensure $i - 1 \geq 0$ and $j - c_i \geq 0$.

Base cases: $C(i, 0) = 0$, $C(0, j) = \infty$.

Runtime: $O(nW)$. Space can be reduced to $O(W)$ if we only store last 2 rows. It is also trivial to store "back-pointers" to recover how we got to $C(i, j)$ using $O(nW)$ space.

1.1.2 Second Solution

Define $C(i)$ to be min # of coins to make i sum. $C(i) = \min\{C(i - c_j) + 1 | \forall c_j\}$. $C(0) = 0$.

This is still $O(nW)$ since for each $1 ... W$, for each $c_1 ... c_n$, we do a constant operation.

1.2 0/1 Knapsack

Given total weight W, values $v_i > 0$, weights $w_i > 0$, find a subset $S \subseteq \{1 ... n\}$ s.t. $\sum_{i \in S} v_i$ is maximized, $\sum w_i \leq W$.

Naive is $O(2^n)$ (try every possible combination of the values).

Solution: let $f(i, j)$ be the maximal value possible if we're given total weight j and objects $1 ... i$. Then $f(i, j) = \max\{f(i-1, j), f(i-1, j - w_i) + v_i\}$. Overall $O(nW)$ time complexity.

Base cases: $f(0, \_) = 0$ (nothing to take), $f(\_, 0) = 0$ (no space to take anything).

1.3 Longest Common Subsequence (LCS)

Def: Given sequences a_i and b_i, find the longest subsequence, indexed at c_i, s.t. $a_{c_i} = b_{c_i}$ (we want to maximize length of c_i).

Naive is something like $O((m+n)2^{m+n})$.

Let $f(i,j)$ be length of longest subsequence for sequences ending at a_i and b_j. Then $f(i,j) = \max\{f(i-1,j-1) + (1 \text{ iff } a_i = b_j \text{ else } 0), f(i-1,j), f(i,j-1)\}$. Evidently, $f(0,\_) = 0, f(\_,0) = 0$.

Runtime is $O(mn)$.

1.4 Sequence Alignment

Given 2 sequences a and b, find an *alignment* (c_i, d_i) s.t. $c_{i-1} < c_i$, $d_{i-1} < d_i$, minimizing cost $\sum_{i=1}^{k} \alpha(a_{c_i}, b_{d_i}) + (m-k)\delta + (n-k)\delta$. Inputs are the α table and δ parameter.

[Needleman-Wunsch '70] Let $f(i,j)$ equals the minimum cost of an alignment that uses the first i values of a, and first j of b. $f(i,0) = i\delta, f(0,j) = j\delta$. Then let $f(i,j) = \min\{f(i-1,j) + \delta, f(i,j-1) + \delta, f(i-1,j-1) + \alpha(i,j)\}$.

1.5 Min-length Triangulation

Def: A polygon P is *convex* if all angles < 180. A *chord* is a line segment between 2 non-adjacent vertices. Problem: given a convex polygon with vertices $v_1 ... v_n v_1$ (in CCW), find a triangulation with minimum total length (of all chords + boundary edges).

Define subproblems: $f(i,j)$ = length of min triangulation for the polygon $v_i ... v_j v_i$.

Base cases: $f(i,i+2) = d(i,i+1) + d(i+1,i+2) + d(i,i+2)$ (d is distance function between 2 vertex indices). $f(i,i+1) = d(i,i+1)$.

$f(i,j) = \min_{k \in i+1 ... j-1}\{f(i,k) + f(k,j) + d(i,j)\}$