CS170 Discussion Section 5 Solutions: *3/8*

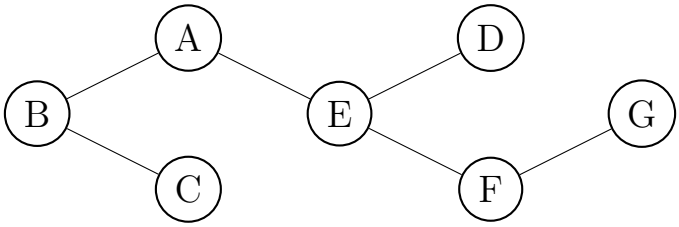
Vertex cover

A *vertex cover* of a graph $G = (V, E)$ is a subset of vertices $S \subseteq V$ that includes at least one endpoint of every edge in E. Give a linear-time algorithm for the following task:

Input: An undirected *tree* $T = (V, E)$.

Output: The size of the smallest vertex cover of T.

For instance, in the following tree, possible vertex covers include $\{A, B, C, D, E, F, G\}$ and $\{A, C, D, F\}$ but not $\{C, E, F\}$. The smallest vertex cover has size 3: $\{B, E, G\}$



Solution

The subproblem $V(u)$ will be defined to be the size of the minimum vertex cover for the subtree rooted at node u. We have $V(u) = 0$ if u is a leaf, as the subtree rooted at u has no edges to cover. The crucial observation is that if a vertex cover does not use a node it has to use all its neighboring nodes. Hence, for any internal node i, we can either use node i, or we do not use node i, and instead use its children:

$$V(i) = \min \left\{ 1 + \sum_{j:(i,j)\in E} V(j), \sum_{j:(i,j)\in E} \left(1 + \sum_{k:(j,k)\in E} V(k) \right) \right\}$$

The algorithm can then solve all the subproblems in order of decreasing depth in the tree and output $V(n)$. Note that any value $V(k)$ is used at most twice - once each while calculating V for the parent and grandparent of k. Thus, the running time is $O(n)$.

Pig

Pig is a 2-player game played with a 6-sided die. On your turn, you can decide either to roll the die or to pass. If you roll the die and get a 1, your turn immediately ends and you get 1 point. If you instead get some other number, it gets added to a running total and your turn continues (i.e. you can again decide whether to roll or pass). If you pass, then you get either 1 point or the running total number of points, whichever is larger, and it becomes your opponent's turn. For example, if you roll 3, 4, 1 you get only 1 point, but if you roll 3, 4, 2 and then decide to pass you get 9 points. The first player to get to 100 points wins.

Suppose at some point the player whose turn it is has x points, their opponent has y points, and the running total for this turn so far is z. Let's work out what the optimal strategy is.

1. Let $W(x, y, z)$ be the probability that the current player will eventually win if both players play optimally. What is $W(x, y, z)$ if $x + z \geq 100$?

   Solution: $W(x, y, z) = 1$, since the current player can just pass, earning z points on top of the x they have already and thus winning immediately.

2. Suppose the current player decides to roll, and gets a 1. What's the probability that they'll still win, in terms of the function W?

   Solution: Let's call the current player A, and their opponent B. After A rolls a 1, it is now B's turn to move, they have y points, their opponent (A) has $x + 1$ points (the x they had before, plus the 1 point from rolling a 1), and the running total is 0. So B will win with probability $W(y, x + 1, 0)$, and A will win with probability $1 - W(y, x + 1, 0)$.

3. Give a recursive formula for $W(x, y, z)$.

   Hint: Work out the probabilities R and P that the current player will win if they decide to roll and pass respectively. Their optimal move is to do whichever of these would give the greatest winning probability, so W will be the maximum of R and P.

   Solution: If the current player rolls and gets a 1, they'll win with the probability computed above. If they roll and get a different value v, then v gets added to the running total and it's still their turn, so they'll win with probability $W(x, y, z + v)$. Since each value of the die has probability $1/6$, this means

   $$R = \frac{1}{6}\left(1 - W(y, x + 1, 0)\right) + \frac{1}{6}\sum_{v=2}^{6} W(x, y, z + v).$$

   If instead they pass, they get $\max(1, z)$ points and it becomes their opponent's turn. So similarly to part (2), we have

   $$P = 1 - W(y, x + \max(1, z), 0).$$

   Finally, $W(x, y, z) = \max(R, P)$, and substituting the expressions above gives a recursive formula for W.

4. Describe a dynamic programming algorithm to compute $W(x, y, z)$. If you needed N points to win instead of 100, what would be the asymptotic runtime of your algorithm?

   Solution: We convert the recursive formula above into a recursive algorithm. The base cases are as indicated in part (1): when a player has a large enough running total to win immediately by passing. Note that in the recursive formula, every recursive term either increases the number of points one of the players has, or increases the running total. So the recursion is guaranteed to terminate. Using memoization to keep track of the values of W we have already computed gives a dynamic programming algorithm.

   To work out the runtime of this algorithm when you need N points to win, notice that we never need to compute $W(x, y, z)$ where any of x, y, or z is $N + 6$ or larger. This is because if either player had that many points they would have already won on the previous turn, and if z was that large the current player could have won before the last die roll by passing. So the algorithm will compute W for at most $(N + 6)^3$ different game positions, and therefore its runtime is $O(N^3)$.

Longest common subsequence

Given two strings $x = x_1x_2 \dots x_n$ and $y = y_1y_2 \dots y_m$, we wish to find out the length of their *longest common subsequence*, that is, the largest k for which there are indices $i_1 < i_2 < \cdots < i_k$ and $j_1 < j_2 < \cdots < j_k$ with $x_{i_1}x_{i_2} \dots x_{i_k} = y_{j_1}y_{j_2} \dots y_{j_k}$. For example, the longest common subsequence of "exponential" and "polynomial" is "ponial" with length 6. Show how to do this in time $O(mn)$.

Solution

The alignment producing the longest common subsequence will have in its last position either a deletion, two characters matched (either equal or different characters) or an insertion.

Subproblems: Define the following subproblems for i, j such that $1 \leq i \leq n$, $1 \leq j \leq m$:

$$L(i, j) = \text{length of longest common subsequence between } x[1, \cdots, i] \text{ and } y[1, \cdots, j]$$

Then, the solution to the original problem will be given by $L(n, m)$.

Algorithm and Recursion: Initialize the dynamic program by setting $L(i, 0) = 0$ and $L(0, j) = 0$ for all i, j. The recursion to compute the values $L(i, j)$ correctly is the following:

$$L(i, j) = \max\{L(i-1, j), L(i, j-1), L(i-1, j-1) + \text{equal}(x_i, y_j)\}$$

where $\text{equal}(x, y)$ is 1 if x and y are the same character and is 0 otherwise.

Running Time: The running time is $O(mn)$ as there are mn subproblems, each of which takes $O(1)$ time.

String shuffling

Let x, y, and z be strings. We want to know if z can be obtained only from x and y by interleaving the characters from x and y such that the characters in x appear in order and the characters in y appear in order. For example, if $x =$ **efficient** and $y =$ **ALGORITHM**, then it is true for $z =$ **effALGiORciIenTHMt**, but false for $z =$ **efficientALGORITHMextraCHARS** (miscellaneous characters), $z =$ **effALGORITHMicien** (missing the final t), and $z =$ **randomString** (obviously wrong). How can we answer this query efficiently? Your answer much be able to efficiently deal with strings such as $x =$ **aaaaaaaaaab** and $y =$ **aaaaaaaac**.

Solution

First, we note that we must have $|z| = |x| + |y|$, so we can assume this. Let

$$S(i,j) = \begin{cases} 1 & \text{if } x[0:i] \text{ and } y[0:j] \text{ can be interleaved to make } z[0:i+j] \\ 0 & \text{otherwise} \end{cases}$$

Then

$$S(i,j) = \max\{(z[i+j] == x[i])S(i-1,j), (z[i+j] == y[j])S(i,j-1)\}$$

Our base cases are $S(i,0) = 1$ if the first i characters of x are the first i characters of z and similarly for y. To get some intuition behind this recurrence relation, simply note that all the recurrence relation is doing is taking the last element of z, and checking to see if its equal to either of the last elements of x or y, and recursively trying to see if the remainder matches the remainder of x or y respectively.