Problem Set 9

All parts are due Sunday, November 24 at 6PM.

Name: Diego Escobedo

Collaborators: Joe, Nicholas Baginski, Noah Lee, John Rao, Mikael Nida

Problem 9-1. Preprocessing: Put all the Coolness, Weight, and Time values into hash tables called C, W, and T respectively.

Subproblems: DP(t,w) = max value of coolness that we get when there is t time left and w weight left in our backpack.

Relate: We have two options. We can either wait in line at a given booth or go home. DP(t,w) = max(max($DP(t-1-T[i])+C[i]$) for i in $[1, n]$, DP(t-1-h, b)). Obviously ,we have to check that the time and the weight are both valid in the 'stand in line' option, otherwise we can only go home. The relation is acyclic because for a given time and weight it depends purely on what we did previously.

Base Cases: DP(i,j) where i or j is less than 0 is negative infinity because you're not allowed to do that. DP(0,j) = 0 and DP(i,0) = 0

Solution: DP(k,b)

Running Time: There are at most b*k problems being done, and at each one, we looked at n different booths. Thus, our algorithm runs in O(nkb), which is pseudo polynomial time because we can make certain inputs arbitrarily large.

Problem 9-2. Preprocessing: We create a hash table keyed by protein markers in P and have all the values be 1 (indicating that it is a valid sequence). This is done in kP time. We can call this table P. The DNA strand remains as S.

Subproblems: DP(i) = max value of protein sequences in A[i:]

Relate: DP(i) = max($DP(i + x) + P[S[i : i + x]]$ for x in $[0, 1, ..., k]$). This works because we are checking where the next place to extend to is. We get a value of 1 from the $P[S[i : i + x]]$ if the strand is actually a strand and 0 otherwise. We are also checking all the possible lengths of the sequence including the single nucleotide sequence but going all the way up to k. The relation is acyclic because for a given element its problems are all after it (none before), meaning there are no cycles.

Base Cases: DP(x) where x is greater than the length of S is just 0, which means we have gone over the limit and it is no longer a valid sequence.

Solution: DP(0)

Running Time: We are doing S subproblems, and in each one we did k^2 work (we are checking k different things and each one takes k time to lookup in the hash table). Further, creating the hash table took kP time. So, in total, this runs in O(k(P + kS)). The running time in this case is polynomial, because even though k would usually make it pseudo-polynomial, in this case k is bounded by S (because we could just ignore any sequences longer than S when hashing because they're obviously not going to generate any value for us). So, this means running time is polynomial.

Problem 9-3. Subproblem: DP(i,j,x) is going to be the minimum required drop height between floors i and j when we have x eggs remaining in our basket.

Relate: DP(i,j,x) = min($h_H + max(DP(i, H-1, x-1), DP(H+1, j, x-1))$) for all floors of height H between i and j. The reasoning behind this is the exact same as the egg drop problem. We also know that the relation is acyclic because i is always increasing or j is always decreasing, meaning we never go back and have cycles.

Base Cases: If i is ever bigger than j, or if we run out of eggs, we return infinity. If i is equal to j, return 0.

Solution: DP(1,n,k)

Runtime: there are k*n*n subproblems, and at each subproblem we do O(n) work, meaning the solution is in total $O(k * n^3)$. This is pseudo polynomial because k can be arbitrarily large.

Problem 9-4. Subproblem: DP(i,j,m,n,t) is going to be True if the player whose turn it is (represented by t, which is 1 if it is player 1's turn, 2 otherwise) can force a win. Player 1 has i and j fingers and player 2 has m and n fingers.

Relation: We have two different cases, depending on whose turn it is. If it is player 1's turn:
DP(i,j,m,n,1) = OR(DP(i,j,m+i,n,2) OR DP(i,j,m,n+i,2) OR DP(i,j,m+j,n,2) OR DP(i,j,m,n+j,2)).
This is assuming that m and n are not 0 (because that's going to represent a disabled hand).
Otherwise, if it is player 2's turn:
DP(i,j,m,n,2) = AND(DP(i+m,j,m,n,1) AND DP(i,j+m,m,n,1) AND DP(i+n,j,m,n,1) AND DP(i,j+n,m,n,1)).
This is assuming that i and j are not 0 (because that's going to represent a disabled hand).
Also, at each step we have to have a checker that determines whether i is greater than a, j than b, m than c, or n than d, in which case we would set that 'hand' to 0 and do the DP again.

Base Cases: DP(i,j,0,0,t) = True as long as i and j are not both 0. DP(0,0,m,n,t) = False as long as m and n are not both 0. These represent the cases where player 1 and player 2 wins, respectively.

Solution: DP(1,1,1,1,1) (because the aliens start with 1 finger in each hand). This is acyclic because the number of fingers is strictly increasing until they reach their maximum in which case it goes to 0 and does not move from there.

Runtime: There are 2abcd subproblems at each step and we do constant work at each, which means this runs in O(abcd). This is an exponential runtime because the number of bits in the input is exponential with respect to the runtime of O(abcd).

Problem 9-5.

(a)

(b) Submit your implementation to `alg.mit.edu`.