CS/ECE 473 Fall 2020 Ryan Prendergast (ryanp4)
Homework 2 Problem 1 Noah Watson (nwatson3)
Lawson (laws)

1.a

Describe an algorithm to find an optimal solution $k_1...k_n$ that runs in time polynomial in k and n.

Solution: We define the function minFuncSum(i,k) which finds the minimum function sum of $T_i...T_n$ subject to the constraint that the non-negative integers $k_i...k_n$ sums to k with the following recurrence:

$$\text{minFuncSum(i,k)} = \begin{cases} T_i(k) & \text{if } i = N \\ min_{j=0}^{k}\{T_i(j) + \text{minFuncSum(i+1, k-j)} & \text{otherwise} \end{cases}$$

We find the minimum function sum of the original problem by calling minFuncSum(1, k).

The above recurrence runs in expornential time. However we see that each iteration minFuncSum(i,k) depends on minFuncSum(i + 1, j) for all possible $j = 0$ to n. We memoize in a 2d, $N * k$ array A with the minimum function sum. We also memoize the value of $0 \leq k_i \leq k$ at which the minimum was found.

$A[i][j]$.sum holds the minimum of the functions $T_i...T_n$ subject to the constraint that $k_i...k_n$ sums to j; if $i = N$, then $A[n][j]$.sum $= T_n(j)$. $A[i][j]$.k holds the value of $0 \leq k_i \leq k$ at which the minimum lies; if $i = N$, then $A[n][j]$.k $= j$

Due to the dependency ordering, we can define an iterative algorithm to fill in this $N * k$ array. We iterate from right to left, from $i = n$ to $i = 1$, filling in the whole column of all possible values of j from $0...k$ at each iteration.

To get the sequence $k_1...k_n$, we start at $A[1][k]$ after filling in the whole array with the above algo. $A[i][k]$.k is k_1, then we go to $A[2][k - k_1]$. $A[2][k - k_1]$.k is k_2. In general, $k_i = A[i][k - \sum_{j=1}^{i-1} k_j]$.

See Algorithm 1 bellow.

Time Complexity As seen in Algorithm 1 bellow, filling in the array requires a nested loop over n and k, and each iteration requires $O(k)$ steps to find the minimum. So, the total runtime is $O(k^2 * N)$.
Space Complexity The space required is dominated by the array A, which is size $O(N * k)$. ■

Algorithm 1 Iterative Min Function Sum Series

for $i = n$ to $i = 1$ **do**
  for $j = 0$ to $j = k$ **do**
    if $i = n$ **then**
      $A[i][j].\text{sum} = F_i(j)$
      $A[i][j].\text{k} = j$
    else
      $A[i][j].\text{sum} = \min_{h=0}^{h=j}\{F_i(h) + A[i+1][j-h]$
      $A[i][j].\text{k} = \text{argmin}_{h=0}^{h=j}\{F_i(h) + A[i+1][j-h]$
    end if
  end for
end for
$j \leftarrow k$
for $i = 1$ to $i = n$ **do**
  $k_i \leftarrow A[i][j].k$
  $j \leftarrow j - k_i$
end for

CS/ECE 473 Fall 2020 Ryan Prendergast (ryanp4)
Homework 2 Problem 1 Noah Watson (nwatson3)
Lawson (laws)

1.b

Describe an algorithm to find the optimal value $\sum_{i=1}^{n} k_i$ that runs in time polynomial in k and n, and uses O(k) space.

Solution: We define an algorithm similar to the above, with the realization that we do not need to have access to the whole array A.

We recognize the recurrence for this problem is the same as in problem (a), and we can solve this problem with the iterative algorithm in problem (a), taking only $A[1][k]$ as our solution. However, this take $O(k * n)$ space.

So, we modify the memoization and iterative algorithm as follows: Instead of saving the $n * k$ array A, we save two k-vectors, $A[k]$ and $B[k]$. They represent the current and the previous columns in the 2d array we used in part (a). This is valid because each subproblem depends only on the subproblem at one index higher, so that is the only subproblem we need to save.

See algorithm 2 bellow. $A[j]$ stores the minimum function sum of the current problem such that the sequence of k's sums to j. $B[j]$ stores the minimum function sum of the subproblem one index above such that the sequence of k's sum to j. Note we do not need to save $A[j]$.k or $B[k]$.k as we only need the sum.

Algorithm 2 Iterative Min Function Sum in k space

for $j = 0$ to $j = k$ **do**
  $B[j] = F_n(j)$
end for
for $i = (n-1)$ to $i = 1$ **do**
  for $j = 0$ to $j = k$ **do**
    $A[j] = \min_{h=0}^{h=j}\{F_i(h) + B[j-h]$
  end for
  $B \leftarrow A$
end for
return A[k]

Time Complexity Same as problem (a), the time complexity is $O(k^2 * n)$.

Space Complexity Since we only save two k-vectors at a time, our total space use is 2*k which is $O(k)$.

■