DUE WED FEB 23, 2022 AT 11.59P VIA GRADESCOPE **H4.PATEL.NISARGS**

You will be graded on correctness, precision, and clarity. Do your best to justify your answers concisely. When we ask for you to devise an algorithm, it is customary to provide pseudo-code in the style that every example in class has been presented. The course staff cannot provide hints on the extra credit problem.

PROBLEM 1 *Sprungli bar*

You are given an $n \times m$ Sprungli chocolate bar. Your goal is to devise an algorithm A that takes as input (n, m) and returns the minimal number of cuts needed to divide the bar into perfect squares of either 1x1, 2x2, 3x3, ..., jxj. With each cut, you can split the bar either horizontally or vertically. For example, $A(2,3) = 2$ because 2x3 $\rightarrow$ (2x2, 2x1) $\rightarrow$ (2x2, 1x1, 1x1). The first cut is "V2" for vertical cut at index 2, and the second cut is "H1" for horizontal at index 1.

1. Notice that no matter the rectangle, it is always possible to make a perfect square in the first cut. Show that this strategy fails. Namely, show an input size for which the strategy of picking the cut which creates the largest box leads to extra cuts in total.
2. Devise a dynamic programming algorithm which determines the minimal number of cuts and outputs the list of cuts.

Ans.

1. The given strategy fails for $(n, m) = (6, 7)$. The largest box initially is 6×6 so it cuts that and 5 additional cuts are required to cut the remaining 1×6 bar. $(6 \times 7) \rightarrow (6 \times 6, 6 \times 1) \rightarrow (6 \times 6, 1 \times 1, 5 \times 1) \rightarrow ... \rightarrow (6 \times 6, 1 \times 1, 1 \times 1, 1 \times 1, 1 \times 1, 1 \times 1, 1 \times 1)$. So it takes 6 cuts with largest box strategy.

   A better solution can be obtained in 4 cuts by doing the following:

   1. V3 for $6 \times 7 \rightarrow (6 \times 3, 6 \times 4)$
   2. H3 for $6 \times 3 \rightarrow (3 \times 3, 3 \times 3, 6 \times 4)$
   3. H4 for $6 \times 4 \rightarrow (3 \times 3, 3 \times 3, 4 \times 4, 2 \times 4)$
   4. V2 for $2 \times 4 \rightarrow (3 \times 3, 3 \times 3, 4 \times 4, 2 \times 2, 2 \times 2)$

2. A horizontal cut at index k will divide the bar of size $n \times m$ into $k \times m$ and $n - k \times m$. Furthermore we can conclude that a cut at k will produce the same dimension bars as cut at $n - k$. Same observation can be made for vertical cuts.

   The optimal answer checks for every horizontal and vertical cuts and finds the cut which minimizes the cuts required for the 2 divided bars. The answer

is 1 plus the minimum cuts required by the bars made by the cut. A square bar does not require further cut.

Based on inputs $n \times m$, we can write a DP equation as:

$$Cuts(n,m) = \begin{cases} 0, \text{ if Square Bar, i.e } n = m, \\ 1 + \min \begin{cases} \min_{k=1}^{\frac{n}{2}} \{Cuts(k,m) + Cuts(n-k,m)\} \\ \min_{k=1}^{\frac{m}{2}} \{Cuts(n,k) + Cuts(n,m-k)\} \end{cases} \end{cases} \quad (1)$$

Memoization can be used by calculcating $Cuts$ in the following order:

$Cuts(1,1)$, $Cuts(1,2)$, ... $Cuts(1,m)$,
$Cuts(2,1)$, $Cuts(2,2)$, ..., $Cuts(2,m)$,
...,
$Cuts(n,1)$, $Cuts(n,2)$, ..., $Cuts(n,m)$

We can store the index and orientation of the cut for each index pair and return the list of cuts required.

Pseudocode for SprungliCuts:

SprungliCuts(n,m)

```
1  for i ← {1,...,n}
2    for j ← {1,...,m}
3      if i = j:
4        Cuts(i,i) = 0
5      else:
6        Compute the min of min_{k=1}^{n/2}{Cuts(k,m) + Cuts(n-k,m)} and
         min_{k=1}^{m/2}{Cuts(n,k) + Cuts(n,m-k)}
7        CutIndex(i,j) ← H,V (depending on orientation of cut k), k for (i,j).
             Eg: V3 for (6,7)
8        Cuts(i,j) ← 1+the minimum from Step-3.
9  ListOfCuts ← GetListOfCuts(n,m,CutIndex)
10 return Best_n
```

GetListOfCuts$(n,m,CutIndex)$

```
1  if CutIndex(n,m) ≠ 0
2    ListOfCuts ← add CutIndex(n,m)
3  Call GetListOfCuts for the 2 bars cut by CutIndex(n,m)
4  return ListOfCuts
```

Time Complexity Analysis:
Line 1 is called n times. Line 2 is called m times. Line 6 computes the minimum in $(n/2 + m/2)$ time. Line 9 computes the list in $Cuts(i,j)$ steps which is $\leq n * m$. All other lines take constant time.

Thus the overall time complexity of SprungliCuts is $\Theta(n * m * (n + m))$.

PROBLEM 2 *Age of War*

We want to play *roughly fair* game in cs5800. You are given an array that holds the weights of n people in the class $W = (w_1, w_2, \ldots, w_n)$. Your goal is to divide n people into two teams such that the total weight of the two teams is as close as possible to equal. Describe such an algorithm and give its running time. The total number of people on each team should differ by at most 1. Assume that M is the maximum weight of a person, i.e., $\forall i, w_i \leq M$. The running time should be $O(n^3 M)$. The output of the algorithm should be the list of people on each team and the difference in weight between the teams.

Ans. We have $W = (w_1, w_2, \ldots, w_n)$. Given this information, we can calculate total weight of class in $\Theta(n)$ time. Let the total weight be T.
We need to divide the class in two teams, A and B.
Let us consider a value $Possible(i, j, k)$ which stores either True or False depending on whether from first i people, we can select any j people in team A such that total weight in team A equals k. If it is True then it is also possible for $i - j$ people in team B have weight $T - k$. So we need to find minimum difference in k and $T - k$ = $|k - (T - k)| = |\mathbf{2k - T}|$.

Let us consider the i^{th} person having weight w_i. If the person is selected in team A, it will contribute w_i weight. So if the current weight(k) is possible, then for first $i - 1$ people and selecting $j - 1$ out of them, weight $k - w_i$ is also possible. If i^{th} person is not selected in team A, then for first $i - 1$ people and selecting j out of them, weight k is also possible.

Thus we can write DP equation for *Possible* as follows:

$$Possible(i, j, k) = \begin{cases} \text{True, if } i = 0, j = 0, k = 0 \\ \text{True, if } i \geq 1 \text{ and } Possible(i-1, j, k) = \text{True,} \\ \text{True, if } i \geq 1, j \geq 1, k \geq w_i \text{ and } Possible(i-1, j-1, k-w_i) = \text{True,} \\ \text{False, otherwise} \end{cases} \tag{2}$$

Once we have calculated for $i = 0..n$, $j = 0..n$, $k = 0..T$, we can find the minimum value of $|2k - T|$ from all the True values of $Possible(n, \lfloor n/2 \rfloor, k)$ for $k = 0..T$. This will divide the partition in almost 2 equal parts as A contain $\lfloor n/2 \rfloor$ people. So B will contain $n - \lfloor n/2 \rfloor$ people. Thus number of people in each team differ maximum by 1. We can use memoization to reduce time complexity in the sequence of $i = 0..n$, $j = 0..n$, $k = 0..T$.
We can also store if current person i is selected in team A or not ($Selected(i, j, k)$) whenever $i \geq 1, j \geq 1, k \geq w_i$ and $Possible(i-1, j-1, k-w_i) = \text{True}$ holds. Using backtracking we can find the list of people in each teams.

Pseudocode for AgeOfWar:

```
AgeOfWar(w(1..n))
 1  T ← sum of all the weights in W.
 2  Possible(0,0,0) = True
 3  for i ← {1,...,n}
 4    for j ← {0,...,n}
 5      for k ← {0,...,T}
 6        Possible(i,j,k) = Possible(i−1,j,k)
 7        if j ≥ 1, k ≥ w_i and Possible(i−1,j−1,k−w_i) = True
 8          Possible(i,j,k) = True
 9          Selected(i,j,k) = True
10  (minDiff, minK) ← (min_{k=0}^{T} |2k − T|, corresponding k)
11  currentA = ⌊n/2⌋
12  for i ← {n,...,1}
13    if Selected(i, currentA, minK) = True
14      A ← add Person i with weight w_i in list A
15      (currentA, minK) ← (currentA − 1, minK − w_i)
16    else
17      B ← add Person i with weight w_i in list B
18  return (reverse(A), reverse(B), minDiff)
```

The algorithm returns the list of persons in each team A and B with the minimum difference of weights between teams $minDiff$.

Time Complexity Analysis:
Line 1 will compute T in $\Theta(n)$. Lines 3, 4, 5 will be called for n, n, T times. Thus computing $Possible(i, j, k)$ will take $\Theta(n^2T)$. Line 12 will be called for n times and line 17 will take n time for reversing the lists.

Thus the overall time complexity of AgeOfWar is $\Theta(n^2T)$. Given that maximum weight of a person is M is the maximum weight of a person, i.e., $\forall i, w_i \leq M$. Thus we have maximum value of total weight $T \leq nM$. Thus the time complexity of AgeOfWar is $O(n^3M)$.

PROBLEM 3 *Mashup*

For simplicity, let x and y be *snippets of music* represented as strings over the alphabet of notes $\{A, B, C, \ldots, G\}$. For example, $x = ABC$ and $y = ADEF$. We say that z is a *loop* of x if z is a prefix of x^k for some integer $k > 0$. For example, $z = ABCABCA$ is a loop of x since it is a prefix of $x^3 = ABCABCABC$.

A song s is a *mashup* of loops z and w if it is an interleaving of z and w. For example, one mashup of the loops $z = ABCABCA$ and $w = ADEFADE$ could be $ABCADEABFADECA$. Given song s and snippets (or hooks) of music x, y, devise an algorithm that determines if s is a mashup of loops of x and y.

Describe a dynamic programming solution to this problem as we have done in class by specifying a variable and then providing an equation that relates that variable to smaller instances of itself. Describe an algorithm based on your equation from above. The algorithm should simply output yes or no. Analyze the running time.

Ans. Let $isMashup(n, m)$ denote if the song S of length$(n + m)$ is a mashup of loops W of length n and Z of length m since length of mashup equals sum of length of both the loops. W and Z are loops of snippets X and Y respectively. We can generalize this for substrings of loops W and M as $isMashup(i, j)$ denoting if substring of S with length $(i + j)$ is a mashup of substring $W(1..i)$ with length i and substring $Z(1..j)$ with length j.

Let us look at the $i + j$th character in $S(1..i + j) = S(i + j)$. If it's a mashup than it will be either the last character of $W(1..i) = W(i)$ or last character of $Z(1..j) = Z(j)$. Thus $isMashup(i, j)$ is true if either $S(i + j) = W(i)$ and $isMashup(i - 1, j)$ is true or $S(i + j) = Z(j)$ and $isMashup(i, j - 1)$ is true.

Since W and Z are loops of X and Y, we have $W(i) = X(i \mod length(X))$ and $Z(j) = Y(j \mod length(Y))$.

Thus, given snippets $X(1..x)$, $Y(1..y)$, and song $S(1..s)$, we can write a DP equation as:

$$isMashup(i, j) = \begin{cases} \text{True, if } X(i \mod x) = S(i + j) \text{ and } isMashup(i - 1, j) = \text{ True,} \\ \text{True, if } Y(j \mod y) = S(i + j) \text{ and } isMashup(i, j - 1) = \text{ True,} \\ \text{False, otherwise} \end{cases} \tag{3}$$

We can find if S(1..s) is mashup of snippets X and Y if any of $isMashup(i, j)$, $i + j = s$, $i \geq 0$, $j \geq 0$ is True. Memoization can be used by calculcating $isMashup$ in the following order:
$isMashup(0, 0)$, $isMashup(0, 1)$, ... $isMashup(0, s)$,
$isMashup(1, 0)$, $isMashup(1, 1)$, ..., $isMashup(1, s - 1)$,
$isMashup(2, 0)$, $isMashup(2, 1)$, ..., $isMashup(2, s - 2)$,
...,
$isMashup(s, 0)$
Since $i + j = s$, we only calculate $isMashup(i, j)$ till $i + j = s$.

Pseudocode for MASHUP:

```
MASHUP(X(1..x), Y(1..y), S(1..s))
1  isMashup(0, 0) ← True
2  for i ← {0, ..., s}
3     for j ← {0, ..., s − i}
4        Calculate isMashup(i, j) based on Eq.(3) if i ≠ 0 and j ≠ 0.
5  for i ← {0, ..., s}
6     return True if isMashup(i, s − i) = True.
7  return False
```

Time Complexity Analysis:
Line 2 is called s times. Line 3 is called $s - i$ times. Thus $isMashup$ will be calculated for $s + (s-1) + (s-2) + ... + 1 = \Theta(s^2)$ times. Line 5 is called s times. All other lines take constant time.

Thus the overall time complexity of MASHUP is $\Theta(s^2)$, where s is the length of the song S.

PROBLEM 4 *Extra Credit: Counting problem (10 points)*

Recall the matrix chain problem that we discussed in class. The input consists of n matricies $A_1, A_2, \ldots, A_n$, and the problem computed the minimum number of operations required to multiply all n matricies. Note: the key issue is that the order in which the matricies were multiplied together changes the number of operations required. Also recall in class we derived a recurrence to count the number of ways to ascend n stairs if you take either 1 or 2 steps at a time.

In this extra credit problem, you will derive a recurrence that counts the number of different ways to multiply n matricies. For example, for 3 matricies, it is easy to see that there are only two ways: $A_1 \cdot (A_2 \cdot A_3)$ or $(A_1 \cdot A_2) \cdot A_3$. Four matricies have 5 ways to choose from. Your answer should be in the form of a recurrence, and be accompanied by an explanation of why the recurrence captures the number of different ways to multiple n matricies.

This problem will develop your skills at dissecting a problem into smaller versions of itself and combining them, except it will be aimed at counting, instead of developing an algorithm.

Ans. Let $M(i, j)$ be the matrix multiplication of matrices A_i to A_j. Thus we have $M(1, n) = M(1, k) * M(k + 1, n)$ for $k = \{1, .., n - 1\}$.
Now let $Count(i)$ be the number of ways we can multiply i matrices. Since each way in $M(1, k)$ can be multiplied with each way in $M(k + 1, n)$, we can multiply $M(1, k) * M(k + 1, n)$ in $Count(k) * Count(n - k)$ ways.

We have $M(1, k) = M(1, 1) * M(2, n) = M(1, 2) * M(3, n) = M(1, 3) * M(4, n) = ... = M(1, n - 1) * M(n, n)$. The first and last equalities have 1 element in one of the matrix $M(1, 1)$ and $M(n, n)$ respectively. Both of these equalities contribute $Count(n - 1)$ ways from remaining matrices. For other equalities, it remains $Count(k) * Count(n - k)$ for $k = 2, .., n - 2$.
$Count(0) = 0$, $Count(1) = 0$ and $Count(2) = 1$ are the base cases. Using this we can formulate a recurrence relation as follows:

$$Count(n) = \begin{cases} 0, \text{ if } n \leq 1 \\ 1, \text{ if } n = 2 \\ 2 * Count(n - 2) + \sum_{k=2}^{n-2} Count(k) * Count(n - k) \end{cases} \tag{4}$$