Name: Daniel Kim
ID: 102353420

**CSCI 3104, Algorithms** **Profs. Chen & Grochow**
**Problem Set 9 – Due Fri Apr 10 11:55pm** **Spring 2020, CU-Boulder**

*Advice 1*: For every problem in this class, you must justify your answer: show how you arrived at it and why it is correct. If there are assumptions you need to make along the way, state those clearly.

*Advice 2*: Informal reasoning is typically insufficient for full credit. Instead, write a logical argument, in the style of a mathematical proof.

**Instructions for submitting your solutions**:

- All submissions must be typed.
- You should submit your work through the **class Canvas page** only.
- You may not need a full page for your solutions; pagebreaks are there to help Gradescope automatically find where each problem is. Even if you do not attempt every problem, please allot at least as many pages per problem (or subproblem) as are allotted in this template.

Quicklinks: 1a 1b 1c 2a **??**

---

Name: Daniel Kim
ID: 102353420

**CSCI 3104, Algorithms** **Profs. Chen & Grochow**
**Problem Set 9 – Due Fri Apr 10 11:55pm** **Spring 2020, CU-Boulder**

1. Could dynamic programming be applied to give an efficient solution to the following problems? Justify your answer in terms of the optimal substructure property on the overlapping sub-problems. Even if you think the answer is yes, you do not need to give an algorithm; the question is *not* "show us how to solve it using DP", it is "give an argument as to whether DP is a reasonable approach to try here."

   (a) List maximum.
   *Input:* List $L$ of numbers.
   *Output:* The maximum element in the list.

   ---

   In the list $L$ of size $n$, we need to recursively compare the value of the last element $L(n)$ with the value of the last element $L(n-1)$ in the rest of the list $L$ with size $n-1$
   We will find out the maximum element in the list when the size of the list is at 1.

   Dynamic programming cannot be applied to given an efficient solution. To begin, since the sub-problem is already stored in a list with constant access time, using the dynamic programming will not going to speed up the process because it uses additional memory to computation time. In addition, there are no revisits to the same problem/element in this recursive algorithm. For example, we first compare $L(n)$ which is the last element in the original list with $L(n-1)$. We choose the maximum element from them and compare them with the next element in the list which is $L(n-2)$. In other words, because the algorithm is keep trying to access new elements in the list, this particular problem does not overlap with each other.

(b) Rod cutting.

*Input:* A list of values $v_1, \ldots, v_n$ for rods of length $1, \ldots, n$, respectively.

*Goal:* Divide a rod of length $n$ into pieces of lengths $\ell_1, \ldots, \ell_k$ ($k$ can vary) to maximize the total value $\sum_{i=1}^{k} v_{\ell_i}$.

*Note:* While this problem is discussed on GeeksForGeeks (and elsewhere), the explanation of optimal substructure on GeeksForGeeks, while not incorrect, is not sufficient explanation to demonstrate mastery of this question.

---

In this situation, we can solve this problem with smaller sizes if we cut the rod of size $n$ into smaller pieces. So after the first cut, we can evaluate the two pieces as independent instances of this rod cutting problem, and we will get the comprehensive optimal solution from the optimal solutions from the two sub problems which maximizes the revenue from each of the pieces. Therefore, this rod cutting problem has optimal substructure, and we can solve them each independently to have the overall optimal solution.

In this particular example, dynamic programming can be applied to give an efficient solution.

The sub problems are overlapping with each other due to the recursive algorithm revisiting the same problem many times.
Let's say that the length of the rod is 4 ($n = 4$), so there are 8 ($2^3$) ways cutting the rod, not cutting the rod at all included. So in each inch, we will have two options which are either to cut it or not.

If the rod length is 4 ($n = 4$), there will be 4 ways to cut it in half.
$(4 + 0, 3 + 1, 2 + 2, 1 + 3)$
If the rod length is 3 ($n = 3$), there will be 3 ways to cut it in half. $(3{+}0, 2{+}1, 1{+}2)$
If the rod length is 2 ($n = 2$), there will be 2 ways to cut it in half. $(2 + 0, 1 + 1)$
If the rod length is 1 ($n = 1$), there will be 1 ways to cut it in half. $(1 + 0)$

From the above, when the length is 3 ($n = 3$), this is also included from the sub problem when the length is 4 ($n = 4$). when the length is 2 ($n = 2$), this is also included from the sub problem when the length is 3 ($n = 3$). Which means that instead of the algorithm computing the same each time, the algorithm is computing all the possibilities of cutting the rod with length 4 ($n = 4$). By doing

this, we can store the value of the visited sub problem for the constant amount of time access. This is called top-down with memorization in dynamic programming.

(c) Graph 3-coloring.

*Input:* A simple, undirected graph $G$.

*Goal:* Decide whether one can assign the colors $\{R, G, B\}$ to the vertices of $G$ in such a way that no two neighbors get the same color.

*Hint:* You may find the notion of critical graph useful.

---

Trivial Algorithm

For this particular problem, we need to use an algorithm which generate all the possibilities of the colors and outputs a composition that fits the given conditions, but the time run for this will be in exponential time. This will color one vertex first and then repeat for every uncolored vertex with colored surroundings that have two different possible colors. In other words, this tree is in n-vertices with exactly $3 \cdot 2^{n-1}$ compositions of 3 colors. Because this runs in exponential time, we can apply dynamic programming to give an efficient solution to the following problem.

In order to find and list all the maximal independent sets, we need to apply an exhaustive search for this particular problem. First, let's say that there is an independent set $R$ of $G$ and for each maximal independent set $R$, we need to check if $G - R$ is 2 colors that can be applied since we set the color set R with an random color.

In the trivial algorithm, this concludes with repeatedly generating the paths that it has been already generated before and going through steps by brute force. However, since we are acknowledging that we are only using maximal independent sets in this dynamic programming, this will prevent generating the same path repeatedly in every step.

Name: Daniel Kim
ID: 102353420



2. Write down the recurrence for the optimal solution for each of the following problems. Justify your answer.

   (a) Social distancing gold-panning. Imagine a river network in which your team can pan for gold, but no two of you can stand in adjacent positions. You have some idea of the expected amount $w(v)$ of gold you will find at each location $v$, but must decide in which locations your team should look.
   *Input:* A rooted tree $T$, with root vertex $r \in V(T)$, and vertex weights $w\colon V(T) \to \mathbb{R}_{\geq 0}$
   Output: A subset of vertices $P \subseteq V(T)$ such that no two vertices in $P$ are adjacent, and maximizing the value $\sum_{v \in P} w(v)$.

   ---

   In this particular problem, there are two cases to be considered since we can't have two vertices in $P$ are adjacent.

   First, we cannot include any of its children because of the condition in the problem. In addition, in the subset of vertices $P$ and by including this vertex, the maximum sum can be calculated by including the value of this vertex along with nodes from its sub tree.
   Second, we can include any of its children in this case. In addition, in the subset of vertices $P$ and by excluding this vertex, the maximum sum can be calculated along with nodes from its sub tree.

   Let's say that $dp(i)$ and starting at vertex i, is the maximum sum of the sub tree. Next, let's also say that $dp(r)$ and starting at rooted vertex r, is the maximum sum of Tree $T$.

   $dp(V) = max(\sum_{i=1}^{n} dp(i), w(V) + (\sum_{i=1}^{n}$sum of dp(j) for all children j of i))

Name: Daniel Kim
ID: 102353420

**CSCI 3104, Algorithms** **Profs. Chen & Grochow**
**Problem Set 9 – Due Fri Apr 10 11:55pm** **Spring 2020, CU-Boulder**

Source: Introducing to Algorithms(3rd)
COLLABORATED WITH:
RUIJIANG MA