CS375 Problem Set 3

Due Nov 11th 10PM ET

A general note for CS375: When writing up your homework, please write neatly and explain your answers clearly, giving all details needed to make your answers easy to understand. Graders may not award full credit to incomplete or illegible solutions.

As a reminder, your solutions to these must be typed. You are encouraged to discuss the problems with your classmates, **but your submission should be your own.** When writing your solutions up, you should not be looking at any other student's submission.

Clear communication **is the point**, on every assignment. In general in CS375, unless explicitly specified otherwise, answers should be accompanied by explanations. Answers without explanations may not receive full credit. Please feel free to ask me any questions about explanations that might come up!

Necessary acknowledgements: many of these questions are reproductions or adaptations of questions of Eric Aaron's devising. Others come from CLRS, Erickson's text, or Kleinberg and Tardös' *Algorithm Design* (which is another fantastic algorithms text that I highly recommend perusing).

1. **Buying Candy on a Budget**: As a teacher preparing for Halloween on a limited budget, you want to maximize the total "candy joy score" of the candies you can afford. You have a budget of B dollars and access to n types of candy. Each candy type i has a price p_i, a joy score j_i, and unfortunately because it's (actually now past) Halloween there's only 1 bag left of each. Determine the combination of candies that maximizes the total joy score without exceeding your budget or the available supply of each candy type.

   To set up the parameters more formally:

   - Let B denote the total budget available.
   - Let n denote the number of candy types.
   - For each candy type i (where $i = 1, 2, \ldots, n$):
     - p_i: the price of the candy of type i.
     - j_i: the joy score of the candy of type i.

   Objective:
   Choose a collection of the candies to maximize the summed joy score, subject to staying within your budget.

   (a) First, consider the following algorithm:

```
BestValue($p$, $j$, $B$):

    Sort the values in $j$ so that $j_1 \geq j_2 \geq ... \geq j_n$
    /** (note that we'll need to simultaneously
      * sort $p$ to keep the corresponding values
      * in the same order) **/

    output = empty list

    for $i$ in range (1, $n$ + 1):
        if $p_i \leq B$:
            add $i$ to the output
            $B$ -= $p_i$

    return output
```

In plain English (as if describing this to a friend without any knowledge of CS), explain how this algorithm chooses which pieces to buy.

(b) Show that the algorithm from (a) is incorrect.

(c) Design an algorithm of $\Theta(n2^n)$ runtime that correctly solves this problem (the correctness of this algorithm should be immediately obvious).

(d) Suppose every price p_i and B are all positive integers. Design a dynamic programming algorithm to solve this problem in a runtime that is polynomial in n and B (score will be based on how minimal this polynomial is along with space efficiency). Your answer should include an explanation of the algorithm's correctness and an analysis of the runtime of the algorithm. Include in your explanation where exactly you are using the assumption that all the p_is and B are integers.

(e) Trace through your above algorithm on the input

- $B = 50$
- $p = [16, 42, 5, 33, 7, 23, 6, 20, 50, 48]$
- $j = [10, 19, 1, 18, 6, 17, 3, 14, 24, 23]$

I'm expecting a grid of values as built by your algorithm (you don't have to do this by hand, my solution just printed it by writing code for the algorithm you do above), along with a trace-back computed (you could highlight the path through the array taken by your algorithm in the backtrack step, maybe give an extra highlight for the steps that add something to an output...)

(f) Suppose instead of the prices and budget being positive integers, I promised all the j_is were positive integers. Can you still come up with a dynamic programming algorithm? If so, what is the runtime now a function of? If not, why?

2. **Pizza Delivery Problems**:

(a) In this first problem, let's imagine we're the pizza company Sonimod and our business offers delivery. Luckily all our orders happen to be placed on the same road we're on: to model this, we'll imagine our input as points on the real line $x_1, ..., x_n$ where our business is at the origin (so any x_i may be less than 0). Every order also has a deadline d_i in time that we have to get the pizza there by in order to get paid. Assume that we only have one driver who always travel at unit speed (so it takes t time to travel t distance) and all deliveries are instantaneous. Our goal is to make every delivery before the deadline. suppose we made deliveries by order of deadline (ie, we sorted the orders by the their deadline and made the deliveries in that order). Show that this algorithm is incorrect.

(b) Next, design an algorithm with runtime $\Theta(n \times n!)$ (its correctness should be obvious) that determines whether you can make all the deliveries before their deadlines.

(c) Next, design a dynamic programming algorithm to determine whether we can make all the deliveries before their deadlines (score will be based on how minimal this runtime is along with space efficiency).

(d) Next, suppose Sonimod hired a second driver and expanded their delivery radius beyond the road they are on: now we have a sequence of deliveries to make, where delivery i has a location p_i and a deadline d_i. Our manager, however, has gotten complaints from some of our perceptive customers who are upset that we aren't delivering by order of initial order placement (which is the same as delivering by order of deadline, since all deadlines are just a fixed amount of time after the original order placement). So we have some new rules:

- Every point must be assigned to a driver
- Each driver must visit the deliveries they've been assigned by order of deadline (earliest first, then next earliest, etc.)

- Our goal is still to make every delivery before its deadline

Design a brute-force algorithm to solve this problem. What is its runtime?

(e) Design a dynamic programming algorithm to solve this problem. In your answer, include an argument of its correctness and an analysis of its runtime (as usual, score will be based upon minimality of time/space usage).

3. **Colby Party** I love a good party, but I'm also sensitive to employer / employee dynamics. So I've charted out the entire employment tree of Colby's faculty and staff: the node at the top of the tree is our president, that node's children are the employees he directly manages, those nodes' children are the employees they directly manage, etc. Additionally, I've also determined how much fun I think each employee would be if I invited them to my party. Specifically, I've made a class `TreeNode` (in say Java) with the following fields:

   - `int partyVal` - how much fun this employee will be at a party. Colby's hiring policies actually require this value to be positive.
   - `TreeNode[] children` - the employees this person manages; this field will never be null, but could be length 0

   Our goal is to determine the best people to invite to the party so that no employee/manager combo are invited. I have not included myself in the tree because I have to be invited, I don't manage anyone, and because my most direct boss (Stephanie) is amazing.

   (a) Suppose I come up with the following function:

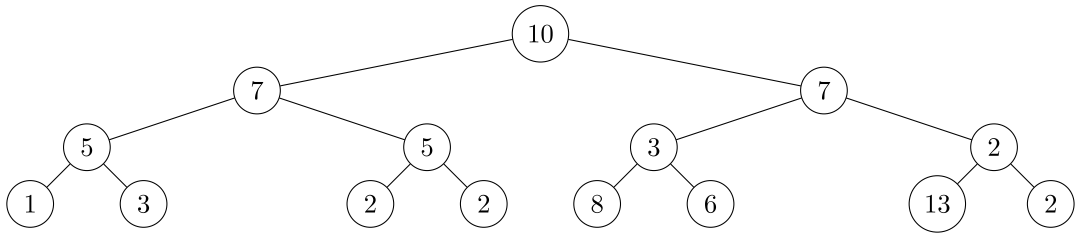
```
bestParty(TreeNode node):
    if (node has no children) return node.partyVal;

    option1 = 0
    for each TreeNode v in node.children:
        option1 += bestParty(v)

    option2 = node.partyVal
    for each TreeNode v in node.children:
        for each TreeNode u in v.children:
            option2 += bestParty(u)

    return max(option1, option2)
```

   Trace through the algorithm on this input:



   Specifically, for each node record the values of option1 and option2 next to it (except for the leaves). Note that although this example is a binary tree, not every input will necessarily be.

4. First, describe in plain English what `option1` and `option2` correspond to.

5. Use your answer in (a) to describe why `bestParty(root)` will return the value of the best party I can make.

6. Why is the runtime of the algorithm in (a) not good?

7. Suppose I allowed you to add fields to the TreeNode class: design and analyze an algorithm for computing `bestParty` in linear time. Explain how your algorithm solves the issue of the pseudocode from (a). I'll give partial credit for explaining why your algorithm is definitely $O(n^2)$.