Homework 3
Sample Solution
Due Date: Thursday, June 7, 11:59 pm

Directions: Your solutions should be typed and submitted as a single pdf on Gradescope by the
due date. LaTeX is preferred but not required. If you use another editor for your solutions (such as
Microsoft Word), you should convert the final document to a pdf, confirm that the symbolic math
from the equation editor is properly formatted, and submit the pdf.

Answers should be clear and concise; show your work but do not write paragraphs where a sentence
suffices. Algorithms should be written in pseudocode or precise sentences that clarify details. Partial
credit will be awarded for incorrect answers that are close and demonstrate understanding of the
problem.

Discussion among students is permitted, but students must write up solutions independently on their
own. In particular, you should understand and be able to explain everything that you write in your
own words. No materials or sources from the Internet (except the course textbook) can be consulted
at any point. Failure to adhere to these policies will result in referral to the Office of Student Conduct.

Problem 1: Greedy Algorithms - Activity Scheduling [20 points]. Suppose you are on vacation
and take a leisurely 5 day cruise. The cruise ship offers a set $S = \{a_1, a_2, \ldots, a_n\}$ of $n$ activities. Each
activity $a_i$ has a start time $s_i$ and a finish time $f_i$. Determined to get your money out of your vacation, you
want to fully complete the most activities possible, but you can only participate in one activity at a time.
Give an algorithm that creates a schedule (a subset of $S$) of activities such that no activities are going on
at the same time, and the total number of such activities is maximized. Argue for the correctness of your
algorithm, and give its running time.

Solution 1. Sort the activities by finish time from least to greatest. Initialize $X = \emptyset$. Let $i^* = \arg\min_{i: a_i \in S} f_i$. Add $a_i$ to $X$, and delete all activities $i$ in $S$ that conflict with $a_i$, that is, the $i \in S$
such that $s_i \leq f_i^{*}$.

We argue correctness by induction on $n$, the number of activities in $S$. In particular, we want to prove
that for all $n$, the predicate $P(n)$, that for any set of $n$ activities, the algorithm returns an optimal schedule,
is true. In the base case of $P(1)$, the algorithm correctly returns the only activity. Make the strong inductive
hypothesis that $P(1), P(2), \ldots, P(n)$ are all true. We want to prove that $P(n + 1)$ is true.

Let $S$ be some set of $n + 1$ activities. Let $a$ be the activity in $S$ with the earliest finish time, and let
$S_a$ be the set of all activities that conflict with $a$, and $S \setminus S_a$ are all other activities. Let $W$ be an optimal
schedule on the set of activities $S$, and let $X$ be the schedule produced by our algorithm. We can partition
$W$ into $W_a$, the activities in $W$ that conflict with $a$, and $W \setminus W_a$, the activities that do not conflict with $a$,
and similarly $X_a$ and $X \setminus X_a$ for our algorithm. Clearly, $|W| = |W_a| + |W \setminus W_a|$, $|X| = |X_a| + |X \setminus X_a|$.

Note that in the first step, our greedy algorithm chooses one activity from $S_a$, and then runs on $S \setminus S_a$.
By the inductive hypothesis, it is optimal on $S \setminus S_a$. Since $W_a \subseteq S_a$, it must be that $|X \setminus X_a| \geq |W \setminus W_a|$. Also, note that $|W_a|$ cannot be more than 1, else there must have been an activity that finished before $a$.
So $|W| \leq |X|$. Therefore $P(n + 1)$ is also true.
Problem 2: Greedy Algorithms - Edge Coloring [20 points]. An edge coloring of a graph is a mapping of every edge to a label (the color) such that no two edges incident on the same vertex have the same label. Let $G = (V, E)$ be an undirected graph, and let $\Delta(G)$ be the maximum degree of any vertex in $G$ (that is, for all vertices $v \in V$, there are no more than $\Delta(G)$ edges incident on $v$). Give a linear time greedy algorithm that produces an edge coloring of a graph using at most $2\Delta(G) - 1$ labels (or colors). Prove that your algorithm never needs more than $2\Delta(G) - 1$ labels (or colors).

Solution 2. Consider the following greedy procedure. We have a number of labels $1, 2, \ldots, 2\Delta(G) - 1$. Consider an arbitrary ordering of the edges. For each edge $e = (u, v)$ in turn, assign it the lowest value label that is not shared by any edges incident on $u$ or $v$. If this process can continue through all of the edges of the graph without needing any more labels, then it clearly gives an edge coloring with no more than $2\Delta(G) - 1$ labels. We prove just that.

Suppose for a contradiction that at some point we encounter an edge $e = (u, v)$ such that $u$ and $v$ have edges with $2\Delta(G) - 1$ different colors already incident on them so that we cannot assign a new color to $(u, v)$. Then one of $u$ or $v$ must have at least $\Delta(G)$ other edges incident on it besides $(u, v)$. But, that means that there is a vertex with degree at least $\Delta(G) + 1$. $\Delta(G)$ is the maximum degree in the graph, so this is a contradiction. Therefore, the greedy process can always assign a label from among $1, 2, \ldots, 2\Delta(G) - 1$.

Problem 3: Dynamic Programming - Word Reconstruction (Taken from DPV 6.4 part a) [20 points]. You are given a string of $n$ characters $s = s[1], \ldots, s[n]$, which you believe to be a corrupted text document in which all punctuation has vanished (so that it looks something like “itwasthestbestoftimes…”). You wish to reconstruct the document using a dictionary, which is given to you as a Boolean function $d(w)$ that takes a string $w$ as input and returns true if $w$ is a valid word and false otherwise. Give a dynamic programming algorithm that determines whether the string $s$ can be reconstituted as a sequence of valid words. Your algorithm should run in $O(n^2)$ time, assuming that calls to $d(w)$ take constant time. Explain why your algorithm is correct.

Solution 3. Let $R(j) = 1$ if the substring $s[1], s[2], \ldots, s[j]$ can be reconstituted as a sequence of valid words. We compute the values in order from $1$ to $n$ as follows.

```
1: procedure RECONSTRUCT(s)
2:   Initialize $R(0) = 1$
3:   for $j = 1$ to $n$ do
4:     Initialize $R(j) = 0$
5:     for $i = j$ down to $1$ do
6:       if $d(s[i], s[i+1], \ldots, s[j]) = \text{true}$ and $R(i-1) = 1$ then
7:         $R(j) = 1$
8:     end if
9:   end for
10: end for
11: end procedure
```

The algorithm has nested for loops over the string, which give the $O(n^2)$ running time. To see that the algorithm is correct, we proceed by induction on $n$, the length of the string. In the base case where $n = 0$, we return true, because the empty string can be reconstituted as a valid sequence of words (namely, the empty sequence). For the inductive case, suppose $R(0), R(1), \ldots, R(n)$ have been correctly computed. We want to prove that $R(n+1)$ is also correctly computed. Simply note that $R(n+1)$ is true just if there is some word that begins at some character $s[i]$ with $i \leq n+1$, ends with $s[n+1]$, and the sequence of characters before $i$ can be reconstituted. Our inner loop checks exactly this condition, since by the inductive hypothesis we have correctly found whether the previous substrings can be reconstituted.
Problem 4: Dynamic Programming - Subset Sum [20 points]. Given a set $X \subset \mathbb{N}$ (that is, a set of natural numbers) with $|X| = n$ (that is, $X$ contains $n$ numbers) and a natural number $t \in \mathbb{N}$, give an algorithm that finds a subset of $X$ such that the sum of numbers in that subset equals $t$, or reports that no such subset exists. Your algorithm should run in $O(nt)$ time. Argue for the correctness of your algorithm.

Solution 4. Let $SS(X, j, k)$ denote the solution subset of $X$ which solves the given problem with the first $j$ elements of set $X$ for a sum of $k$. Then, if there exists a subset of $\{X_1, ..., X_j\}$ such that the subset sums to $k$, $SS(X, j, k)$ will be one such subset. Let $SS(X, j, k) = \text{nil}$ mean that no such subset exists.

```
1: procedure SubsetSum(X, t)
2:   for $j \in \{0, 1, \ldots n\}$ initialize $SS(X, j, 0)$ to \emptyset and initialize $SS(X, j, k)$ to nil for all $1 \leq k \leq t$
3:   for $k$ from 1 to $t$ do
4:     for $j$ from 1 to $n$ do
5:       if $SS(X, j - 1, k) \neq \text{nil}$ then
6:         $SS(X, j, k) \leftarrow SS(X, j - 1, k)$
7:       end if
8:       if $SS(X, j - 1, k - X_j) \neq \text{nil}$ then
9:         $SS(X, j, k) \leftarrow (SS(X, j - 1, k - X_j) \cup X_j)$
10:      end if
11:     if $X_j = k$ then
12:        $SS(X, j, k) \leftarrow \{X_j\}$
13:     end if
14:   end for
15:   return $SS(X, n, t)$
16: end procedure
```

It is easy to see that the algorithm runs in $O(nt)$ time. At each step of iteration it checks three boolean conditions and conditionally performs one boolean assignment, all in constant time, and there are precisely $nt$ iterations. I will prove correctness by induction on $k$ from 0 to $t$ to show that for each value of $k$, the algorithm determines a valid subset summing to $k$ if one exists and yields nil if not. In the base case of $k = 0$, the subset $\emptyset \subseteq X$ is a valid solution for all $j$ from 0 to $n$.

In the inductive step, suppose for some $1 \leq k \leq t - 1$ that for all $k' < k$ the algorithm has correctly determined whether a subset of $X$ exists that sums to $k'$ denoted by $SS(X, j, k')$ for some $j \in \{0, \ldots, n\}$. Then if there is a subset $SS(X, j, k)$ it must be that for some $X_j \in X$ either $X_j = k$ or there was some smaller sum equal to $k - X_j$ not using $X_j$. But since $k - X_j < k$ then by assumption the algorithm has already found such a value should it exist, and by the second condition of the algorithm, the correct corresponding set will be recorded taking the union of that previous set with $X_j$. If it was that $X_j = k$ then the third condition of the algorithm will correctly record the set containing just $X_j$ as a valid solution. Thus, if a valid solution subset exists for $k$, the algorithm finds it.