EECS 340 Algorithms

2018 Spring Semester

Homework 5

Due on April 11th, 12:45pm

This assignment covers dynamic programming and greedy algorithms.

1. **Warm-Up: Applying Dynamic Programming Algorithms** Use the dynamic programming algorithms given in class to solve the following problems. Show your work.

   (a) **Coins in a Line**
   Solve R-12.4 in the text. ($\star$)

   (b) **0-1 Knapsack**
   Solve R-12.5 in the text. ($\star$)

   (c) **Telescope Scheduling Redux**
   Solve R-12.6 in the text. ($\star$)

2. **Warm-Up: Applying Greedy Algorithms** Use the greedy algorithms given in class to solve the following problems. Show your work.

   (a) **Fractional Knapsack**
   Solve R-10.1 in the text. ($\star$)

   (b) **Task Scheduling**
   Solve R-10.3 in the text. ($\star$)

3. **Warm-Up: Coins in a Line and the Failings of Greed**

   Solve C-12.3 in the text. ($\star\star$)

4. **The Cleveland Pothole Problem**

   Brick roads evidently not being bumpy enough, Little Italy's Mayfield Road has recently developed several potholes. The potholes this year are very bizarre – they are zero-dimensional points which send cars passing over them to a place nobody

knows. [1] Unfortunately, the city does not have enough money to hire construction workers, but fortunately, the city does have an enormous reserve supply of 24'x24' [2] metal covering plates to prevent cars from entering the underworld. Suppose that you are given an array of the positions of these potholes [3] $\{s_1, ...s_n\}$ where each s_i represents the distance along the road from the intersection of Euclid and Mayfield to a pothole. Give an algorithm which returns the minimum number of metal plates required to cover all potholes, and prove its correctness. $(\star\star)$

5. **Zombie Apocalypse: Cooking Edition**

   The year is 2050, and you're the manager for Bone Apple Tea, CWRU's only dining service left operating after the start of this year's HvZ. You have been asked by Babs to distribute rations to maximize the number of students who survive. Luckily, you have exactly as many packs of rations as there are students, but your entire kitchen has been zombified and the packs of rations are of wildly varying caloric values. Suppose that you have a list $C = (c_1, ..., c_n)$ of the calorie counts of the rations, and a list $B = (b_1, ..., b_n)$ of the benefits [4] per calorie toward survivability of each student [5]. A *pairing* of students with rations is an assignment of one and only one ration [6] to each student, represented as a list of pairs of the form (i, j) where i represents the index of a ration in the list of rations, and j represent the index of the student it was assigned to in the list of the students' benefits per calorie. Give an algorithm to pair students and rations so as to maximize the total benefit toward survivability, and prove its correctness. $(\star\star)$

6. **Checker-Sum Redux**

   Recall from Homework 2 that for a matrix $A \in \mathbb{R}^{n \times n}$, we call the *checkerboard sum matrix* of A the matrix whose ith column and jth row is given by:

$$\texttt{checker-sum}(A)_{i,j} = (-1)^{i+j} \sum_{k=1}^{i} \sum_{l=1}^{j} (-1)^{k+l} A_{k,l}$$

   Use dynamic programming to implement $\texttt{checker-sum}(A)$. $(\star\star)$

7. **On The Grid**

   Suppose that we have a recursive function $F : \{1, ...n\} \times \{1, ...m\} \to \mathbb{N}$ implemented

[1] The length of the potholes being zero in Cleveland is how you know that this question is purely hypothetical.

[2] Enough to stretch the entire width of the road.

[3] In no particular order.

[4] In *Utils*, a completely fake unit also used for keeping the business students happy.

[5] That is, student i receives the benefit b_i per calorie.

[6] Company policy only allows one portable rations swipe per person per week.

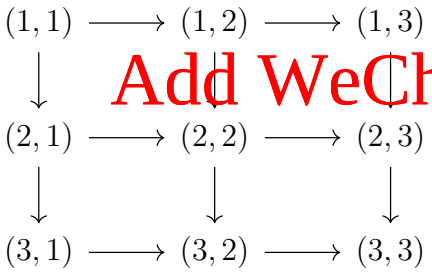
as:

Algorithm 1: F(n, m)

```
if Test(n, m) then
    return Base(n, m)
end
else
    I ← NextIndices(n, m);
    Results ← [] ;
    for (i, j) ∈ I do
        Results.append(F(i, j))
    end
    return Combiner(Results)
end
```

For some implementations of the functions "Base", "Combiner", and "NextIndices" which you do not have access to the implementations of.

Suppose that we draw a $n \times m$ grid, where every grid point corresponds naturally [7] to a point in $\{1, \ldots, n\} \times \{1, \ldots, m\}$. Consider these grid points as vertices of a directed graph $G = (V, E)$, where there is an edge from (i, j) to (k, l) iff any call to $F(k, l)$ directly results in a recursive call to $F(i, j)$ [8] For example, if $m = n = 3$, and $F(i, j)$ depends on $F(i - 1, j)$ and $F(i, j - 1)$ when i and j are both not 1, this graph will look like:



Describe a dynamic programming algorithm whose inputs are m, n and a topological ordering of the graph G (constructed as above) which returns the value of $F(n, m)$ $(\star\star)$.

8. **A State of Balance**

   Solve C-12.13 in the text. $(\star\star)$

9. **Monotonically Decreasing Subsequences**

   Describe an $O(n^2)$ algorithm to find the length of the longest monotonically (strictly) decreasing subsequence(s) of n integers. For example, given the sequence $(2, 4, 1, 3)$

[7] $(Row, Column)$, where row numbers increase downward and column numbers increase rightward.
[8] That is, (i, j) is in the list returned by NextIndices(k, l).

$(n = 4)$, the decreasing subsequences are $(4, 1)$, $(2, 1)$, $(4, 3)$, (2), (4), (1), and (3), so your algorithm should return 2 in this case. $(\star\ \star\ \star)$

10. **Let's Do the Time Warp Again**

    Solve A-12.8 in the text. $(\star\ \star\ \star)$

11. **The following is for EECS 454 students only**. Read Section 17.4 in the textbook and then solve the following problems:

    (a) R-17.9

    (b) R-17.10

    (c) R-17.12

    (d) C-17.10

Submission

Hand in your paper in-class by the beginning of class on the due date. Always check Canvas for updates and corrections.