CS5800 Algorithms *Out: 19 October 2021*

Problem Set 4

Ravi Sundaram *Due: 29 October 2021*

Dynamic Programming Solution Guidelines

Dynamic programming can be very tricky and this template will help guide you through solving new problems. Get in the habit of going through the list and filling everything out step by step. We will grade harshly on missing items. And if there's no english description of the two items mentioned below then the solution will get an **AUTOMATIC 0** on homeworks and exams with no exceptions.

1. (**AUTOMATIC 0 IF MISSING**) English description of the
   (a) recursion you use/the memoization table.
   (b) logic behind your recursion.
2. The actual recursion. And don't forget the base cases!
3. Final value we have to return.
4. Brief description of how an iterative algorithm would loop through the memoization array and fill the values (make sure your order makes sense, and that values are already filled when you call them). Pseudocode isn't required, just a couple sentences.
5. The number of subproblems you have to solve.
6. How much time each subproblem takes to solve (usually constant or linear).
7. Final running time.

Problem 1 (Alice and Bob and Alternating Games) 25

Suppose Alice and Bob are given an array $A[1...n]$ of integers. They are playing a game where they alternate turns, and at each turn a player chooses one of the two integers at the end of the array. After they choose that number, the number gets deleted from the array and it's the next person's turn. The winner of the game is the one who has the larger sum from the numbers they have chosen. Give a dynamic programming solution for Alice to *maximize the sum of the integers she chooses*, assuming that Bob plays optimally.

For example, suppose we have the array $A = [8, 7, 8, 9]$. Alice can choose 9 in the first turn. Then we have the array $[8, 7, 8]$ and Bob can choose either 8. Then Alice has a choice between picking 7 and 8 so she chooses 8 which leaves Bob with 7. Alternatively, if Alice had initially chosen 8, then Bob would have chosen the 9 and Alice would choose the remaining 8. Thus, for this specific configuration, with optimal play, Alice and Bob will both have the same totals.

Solution:

1. The recursion to solve this problem is that for the remaining cards $A[i..j]$, the score Alice could get, denoted by $OPT(i, j)$, is the maximum of the two situations: 1) she picks the first card i, and since Bob plays optimally he would pick from the cards $A[i + 1..j]$ such that Alice has the lower score, i.e., Alice would be left with the worse case of $A[i + 2..j]$ and $A[i + 1..j - 1]$ after

Bob picked. Thus, $OPT(i,j) = A[i] + \min\{OPT(i+2,j), OPT(i+1,j-1)\}$; 2) she picks the last card, similarly, $OPT(i,j) = A[j] + \min\{OPT(i+1,j-1), OPT(i,j-2)\}$.

2. $$OPT(i,j) = \begin{cases} 0 & \text{if} \quad i > j \\ A[i] & \text{if} \quad i = j \\ \max\left\{ \begin{array}{l} A[i] + \min\{OPT(i+2,j), OPT(i+1,j-1)\}, \\ A[j] + \min\{OPT(i+1,j-1), OPT(i,j-2)\} \end{array} \right\} & \text{if} \quad i < j \end{cases}$$

3. The final value to return is $OPT(1,n)$

4. The iterative approach would fill out an $n \times n$ matrix and the base case if the diagonal where $i = j$. The area below the diagonal is 0, while the upper triangle can be filled out using the recursion described above.

5. The number of subproblems is $\theta(n^2)$ as we have to fill out half of the $n \times n$ matrix.

6. Each subproblem takes constant time, i.e. $\theta(1)$.

7. Total running time is $\theta(n^2)$.

■

Problem 2 (Ternary Tree Track Totals) 25

A ternary tree a rooted tree where each node (except the leaves) have three children each. We are given a ternary tree T with a positive integer label on each node of the tree. Further, you are given that the tree has k levels such that at level $i \in \{2, \ldots, k\}$, there are 3^{i-1} nodes since every node at level $i-1$ has 3 children each and one node at the root.

You want to find the maximum path sum starting at the root of the tree and following any path on the tree from root to a leaf. From every node in your path, except the terminal leaf node, you have three options for which child to use for your path.

Solution:

1. The recursion for this problem is that for each node the maximum path starting from the current node is the one that contains the maximum path from its children. The algorithm terminates if the node has no children.

2. For each node p in tree T:

$$OPT(p) = \begin{cases} p.val & \text{if } p.child = \emptyset \\ p.val + \max\limits_{c \in p.child} OPT(c) & \text{otherwise} \end{cases}$$

3. The final value is returned by $OPT(\text{root})$

4. The iteration approach need to fill out a $k \times 3^{k-1}$ table. The first row stores the label integer of each node at the last level. Then for the rest rows, each cell can be calculated by looking at the 3 cells on the previous row corresponding to its children node.

5. The number of subproblems is exponential.

6. Each subproblem takes constant time, i.e., $\theta(1)$.

7. Total running time is exponential.

■

Problem 3 (Counting Coin Changes) 25

Given a denomination of positive coins $c_1, c_2, \ldots, c_m$ and a value n as input how many ways can you make change for n. For example, with coins being $\{1, 2, 3\}$ the number of ways to get the value 4 is 4 as follows: $\{1, 1, 1, 1\}, \{1, 1, 2\}, \{2, 2\}, \{3, 1\}$. With coins being $\{2, 5, 3, 6\}$ the number of ways to get the value 10 is 5 as follows: $\{2, 2, 2, 2, 2\}, \{2, 2, 3, 3\}, \{2, 2, 6\}, \{2, 3, 5\}, \{5, 5\}$.

Solution:

1. Similar to the Knapsack problem, the recursion can be established by including the parameters m and n, where m indicates that the set of m coins with denominations of $c_1, ..., c_m$ and n is the amount to make change for. For an amount n, the total ways to make change using the set of coins $c_1, ..., c_m$ is $T(m, n) = T(m, n - c_m) + T(m - 1, n)$. The first part corresponds to the case that the coin with largest value c_m is used and the second part for not using c_m.
2. $$T(m,n) = \begin{cases} 1 & \text{if } n = 0 \\ T(m-1, n) & \text{if } c_m > n \\ T(m, n - c_m) + T(m-1, n) & \text{otherwise} \end{cases}$$
3. The final value is returned by $T(m, n)$.
4. The iteration can be implemented with a memoization table of $m \times n + 1$ and the base case is $T(i, 0) = 1$ for $i = 1..m$. The rest of the cells can be filled out using the recursion above.
5. The number of subproblems is $\theta(mn)$.
6. Each subproblem takes time of $\theta(1)$.
7. Total running time is $\theta(mn)$.

■

Problem 4 (Programming: Dwarf Dormitories) 25

Problem Statement: You are Durin, the oldest of the dwarves, and are looking for a suitable place for your people to live. After much searching, you have found Khazad-dûm, a series of caves and tunnels under the mountains where you and all of your dwarves can live happily.

You have carved out a deep tunnel of low width, and used to stone to create a series of rooms inside (some higher quality than others), in the form of an R x C grid (R rows, C columns, where C is a small number). Each room has a quality measure q_{ij}, indicating how happy a dwarf would be to live in the room in the i^{th} row and j^{th} column. You would like to place the dwarves in such a way as to maximize the sum of their happiness values.

You have exactly $R * C$ dwarves to house, and had thought that you would have enough rooms. However, there is a problem. Dwarves are private creatures, and none of your dwarves will take a room if it borders a room shared by another dwarf horizontally or vertically (diagonally adjacent rooms are okay). Given this constraint, and the quality values of each room, figure out the maximum possible happiness sum. There will be left over dwarves which do not have a room in the grid - they do not contribute to the score.

Input Format: Line 1: R C, indicating that the grid has R rows and C columns Next R lines: A space delimited list of C integers, indicating the quality values of the rooms in the corresponding row. The rows are given in order from the first row to the R^{th} row.

Constraints: $1 \leq R \leq 10^5$ $1 \leq C \leq 6$ $0 \leq q_{ij} \leq 100$

Output Format: Line 1: S, the maximum possible happiness sum achievable by playing dwarves in the grid of rooms, subject to the constraints in the problem statement. Your output should end in a new line.

Sample Input: 2 2 5 0 10 6

Sample Output: 11

Explanation: The max possible happiness sum is to put one dwarf in the room of quality 5, and another in the room of quality 6.