CS 170 HW 7

Due **2021-3-9, at 10:00 pm**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, write "none".

DP solution writing guidelines:

Try to follow the following 3-part template when writing your solutions.

- Define a function $f(\cdot)$ in words, including how many parameters are and what they mean, and tell us what inputs you feed into f to get the answer to your problem.
- Write the "base cases" along with a recurrence relation for f.
- Prove that the recurrence correctly solves the problem.
- Analyze the runtime and space complexity of your final DP algorithm.

2 Counting Targets

We call a sequence of n integers $x_1, \ldots, x_n$ *valid* if each x_i is in $\{1, \ldots, m\}$.

(a) Give a dynamic programming-based algorithm that takes in n, m and "target" T as input and outputs the number of distinct valid sequences such that $x_1 + \cdots + x_n = T$. Your algorithm should run in time $O(m^2 n^2)$.

(b) Give an algorithm for the problem in part (a) that runs in time $O(mn^2)$.
Hint: let $f(s, i)$ denotes the number of length-i valid sequences with sum equal to s. Consider defining the function $g(s, i) := \sum_{t=1}^{s} f(t, i)$.

Solution:

(a) We use $f(i, s)$ to denote the number of sequences of length i with sum s. $f(s, i)$ is 0 when $i > 0$ and $s \leq 0$, and $f(s, 1)$ is 1 if $1 \leq s \leq m$. Otherwise it satisfies the recurrence:

$$f(s, i) = \sum_{j=1}^{m} f(s - j, i - 1)$$

There are a total of mn^2 subproblems and it takes $O(m)$ time to compute $f(s, i)$ from its subproblems, which leads to an $O(m^2 n^2)$ DP algorithm. Our algorithm outputs $f(T, n)$.

(b) We define $g(s,i)$ as follows:

$$g(s,i) = \sum_{j=1}^{s} f(j,i)$$

This is equal to

$$\begin{aligned} g(s,i) &= f(s,i) + \sum_{j=1}^{s-1} f(j,i) \\ &= \sum_{j=1}^{m} f(s-j,i-1) + g(s-1,i) \\ &= g(s-1,i-1) - g(s-m-1,i-1) + g(s-1,i). \end{aligned}$$

Using this recurrence, there are still mn^2 subproblems, but it takes $O(1)$ time to compute $g(s,i)$ from its subproblems, and thus there is a $O(mn^2)$ time DP algorithm. We can then obtain $f(T,n)$ via $g(T,n) - g(T-1,n)$.

3 Knightmare

Give an algorithm to find the number of ways you can place knights on an N by M ($M < N$) chessboard such that no two knights can attack each other (there can be any number of knights on the board, including zero knights). Clearly describe your algorithm and prove its correctness. The runtime should be $O(2^{3M} M \cdot N)$.

(Please provide a 3-part solution)

Solution:

We use length M bit strings to represent the configuration of rows of the chessboard (1 means there is knight and otherwise 0).

The main idea of the algorithm is as follows: we solve the subproblem of the number of valid configurations of $(n-1) \times M$ chessboard and use it to solve the $n \times M$ case. Note that as we iteratively incrementing n, a knight in the n-th rows can only affect configurations of rows $n+1$ and $n+2$. So we can denote $K(n,u,v)$ as the number of possible configurations of the first n rows with u being the $(n-1)$-th row and v being the n-th row, and then use dynamic programming to solve this problem.

Pseudocode: We say bit strings u, v is valid no two knights can attack each other in the $2 \times M$ table represented by u and v. Similarly we say bit strings u, v, w is valid no two knights can attack each other in the $3 \times M$ table represented by u, v and w.

```
Initialize $K(\cdot,\cdot,\cdot) := 0$
for all size $M$ bitstrings $v, w$ do
    Initialize $K(2, v, w) := 1$ if $v, w$ is valid else 0
for $n = 3$ to $N$ do
    for all size $M$ bitstrings $u, v, w$ if $u, v, w$ is valid do
        $K(n, v, w) += K(n-1, u, v)$
    return $\sum_{v,w} K(N, v, w)$
```

Proof of Correctness: By definition, $K(2, v, w) = 1$ if the M by 2 chessboard configuration defined by v and w is legitimate. Otherwise, $K(2, v, w) = 0$.

For $n > 2$, we have

$$K(n, v, w) = \sum_u K(n - 1, u, v),$$

where we are summing over all possible configurations u for the third-last row of a chessboard whose last rows are specified by v and w.

To bound the total runtime, first note that for each row, we iterate over all possible configurations of knights in the row and for each such configuration, we perform a sum over all possible valid configurations of the previous two rows. The time to check if a configuration is valid is $O(M)$. Therefore, the time taken to compute the sub-problems for a single row is $O(2^{3M}M)$ which gives us an overall runtime of $O(2^{3M}MN)$.

4 Geometric Knapsack

Suppose we a rectangular paper of side lengths X, Y, where X, Y are positive integers, and a set of n *products* that can be made of the paper. Each product is a rectangle of dimensions $a_i \times b_i$ and of value c_i, where all these numbers are positive integers. Suppose we can only cut the paper horizontally and vertically.

Describe and analyze an algorithm that determines the maximum value of the products that can be made out of the single $X \times Y$ paper. You may produce a product multiple times, or not at all if you wish.

Solution: For $1 \le i \le X$ and $1 \le j \le Y$, let $C(i, j)$ be the best return that can be obtained from a cloth of shape $i \times j$. Define also a function `rect` as follows:

$$\texttt{rect}(i, j) = \begin{cases} \max_k c_k & \text{over all products } k \text{ with } a_k = i \text{ and } b_k = j \\ 0 & \text{if no such product exists} \end{cases}$$

Then the recurrence relation is

$$C(i, j) = \max \left\{ \max_{1 \le k < i} \{C(k, j) + C(i - k, j)\}, \max_{1 \le h < j} \{C(i, h) + C(i, j - h)\}, \texttt{rect}(i, j) \right\}$$

It remains to initialize the smallest subproblems correctly:

$$C(1, j) = \max\{0, \texttt{rect}(1, j)\}$$
$$C(i, 1) = \max\{0, \texttt{rect}(i, 1)\}$$

The final solution is then the value of $C(X, Y)$.

To implement the algorithm, we simply initialize a $X \times Y$ 2D array and set up the base cases as above. Then we fill the DP table row by row from the top, starting from $C(2, 2)$ and applying the recurrence relation for each entry.

For proving correctness, notice that $C(i, j)$ trivially has the intended meaning for the base cases with $i = 1$ or $j = 1$. Inductively, $C(i, j)$ is solved correctly, as a rectangle $i \times j$ can only be cut in the $(i - 1) + (j - 1)$ ways considered by the recursion or be occupied completely by a product, which is accounted for by the `rect`(i, j) term. The running time is $O(XY(X + Y + n))$ as there are XY subproblems and each takes $O(X + Y + n)$ to evaluate.

5 GCD annihilation

Let $x_1, \dots, x_n$ be a list of positive integers given to us as input. We repeat the following procedure until there are only two elements left in the list:
Choose an element x_i in $\{x_2, \dots, x_{n-1}\}$ and delete it from the list at a cost equal to the greatest common divisor of the undeleted left and right neighbors of x_i.

We wish to make our choices in the above procedure so that the total cost incurred is minimized. Give a poly(n)-time dynamic programming-based algorithm that takes in the list $x_1, \dots, x_n$ as input and produces the value of the minimum possible cost as output. You may assume that we are given an $n \times n$ sized array where the i, j entry contains the GCD of x_i and x_j, i.e., you may assume you have constant time access to the GCDs.

Solution: Let $F(a, b)$ be the minimum cost incurred when the input is the subarray between indices a and b. When $b = a + 1$, $F(a, b) = 0$. Suppose in performing the deletion on the $[a, b]$ subarray, element s is the last element to be deleted, the total cost incurred is equal to $F(a, s) + F(s, b) + \gcd(x_a, x_b)$. This tells us that when $b > a + 1$,

$$F(a,b) = \min_{a+1 \le s \le b-1} F(a,s) + F(s,b) + \gcd(x_a, x_b)$$

Thus, if we turn the above recurrence to a DP algorithm, we get an $O(n^3)$ time algorithm since computing $F(a, b)$ from its subproblems takes up to $O(n)$ time and there are a total of $O(n^2)$ subproblems. The output of our algorithm is $F(1, n)$.