Ravi Dayabhai

Problem Session 9

Problem 9-1. Coin Crafting

Ceal Naffrey is a thief in desperate need of money. He recently acquired n identical gold coins. Each coin has distinctive markings that would easily identify them as stolen if sold. However, using his amateur craftsman skills, Cael can melt down gold coins to craft other golden objects. Ceal has a buyer willing to purchase golden objects at different rates, but will only purchase one of any object. Ceal has compiled a list of the n golden objects, listing both the positive integer **purchase price** the buyer would be willing to pay for each object and each object's positive integer **melting number**: the number of gold coins that would need to be melted to craft that object. Given this list, describe an efficient algorithm to determine the maximum revenue that Ceal could make, by melting down his coins to craft into golden objects to sell to his buyer.

Problem 9-2. Career Fair Optimization

Tim the Beaver always attends the career fair, not to find a career, but to collect free swag. There are n booths at the career fair, each giving out one known type of swag. To collect a single piece of swag from booth i, having integer coolness c_i and integer weight w_i, requires standing in line at that booth for integer t_i minutes. After obtaining a piece of swag from one booth, it will take Tim exactly 1 minute to get back in line at the same booth or any other. Tim's backpack can hold at most weight b in swag; but at any time Tim may spend integer h minutes to run home, empty the backpack, and return to the fair, taking 1 additional minute to get back in a line. Given that the career fair lasts exactly k minutes, describe an $O(nbk)$-time algorithm to determine the maximum total coolness of swag Tim can collect during the career fair.

Problem 9-3. Protein Parsing

Prof. Leric Ander's lab performs experiments on DNA. After experimenting on any **strand of DNA** (a sequence of nucleotides, either `A`, `C`, `G`, or `T`), the lab will cut it up so that any useful protein markers can be used in future experiments. Ander's lab has compiled a list P of known protein markers, where each **protein marker** corresponds to a sequence of at most k nucleotides. A **division** of a DNA strand S is an ordered sequence $D = (d_1, \ldots, d_m)$ of DNA strands, where the ordered concatenation of D results in S. The **value** of a division D is the number of DNA strands in D that appear as protein markers in P. Given a DNA strand S and set of protein markers P, describe an $O(k(|P| + k|S|))$-time algorithm to determine the maximum value of any division of S.

Problem 9-4. Lazy Egg Drop

The classic egg drop problem asks for the minimum number of drops needed to determine the breaking floor of a building with n floors using at most k eggs, where the **breaking floor** is the lowest floor from which an egg could be dropped and break. This problem has a closed form solution, but can also be solved with dynamic programming (Exercise!). However, if the building does not have an elevator, one might instead want to minimize the **total drop height**: the sum of heights from which eggs are dropped. Suppose each of the n floors of the building has a known positive integer height h_i, where floor heights strictly increase with i. Given these heights, describe an $O(n^3k)$-time algorithm to return the minimum total drop height required to determine the breaking floor of the building using at most k eggs.

① This seems like a lightly reskinned version of the 0/1 Knapsack problem.

S Let $x(i,j)$ be the revenue maximizing over i remaining wins and objects indexed $\{1,\dots,j\}$ where $1 \le j \le n$.

R $x(i,j) := \max\{$
$P[j] + x(i - M[j], j-1)$ if $i \ge M[j]$ else 0,
$x(i, j-1)\}$

I decreasing j

B $x(0,j) = 0$ for all $j \in \{1,\dots,n\}$

O $x(n,n)$

T There are $O(n^2)$ possible states wherein $O(1)$ non-recursive work determines any state $\Rightarrow O(n^2)$ overall run time.

② **S** Let $x(i,j)$ be the max. coolness possible when i minutes are left at the fair and Tim's backpack has j capacity left.

R $x(i,j) := \max\{\{0\} \cup$

$\{c_\ell + x(i-1, j-w_\ell) \mid 1 \le \ell \le n,\ j \ge w_\ell,\ i \ge 1\} \cup$

$\{c_\ell + x(i-b, -1, b-w_\ell) \mid 1 \le \ell \le n,\ b \ge w_\ell,\ i > b\}$

$\}$ where w_ℓ, c_ℓ for $\ell \in \{1, \ldots, n\}$ are the respective weights & coolness of swag.

I increasing i, j

B $x(0,j) := 0$ for all $0 \le j \le b$
$x(i,0) := 0$ for all $i \le h$

O $x(k,b)$

T $O(nbk)$ time since there are $O(bk)$ states, and computing each state takes $O(n)$ work.

7

S Let $x(i)$ be the maximum value of dividing $S[i:]$.

R $x(i) := \max\{$

$(1 \text{ if } s[i:j] \in P \text{ else } 0) + x(j)$

$\mid \; i \le i < j \le \min(i+k, |S|)\}$ $O(k)$

almost! as written, hashing each $S[i:j]$ is $O(k^2)$ work, but if we only look-up the $O(k)$ strings (all starting from i) it's $O(k)$ work!

T decreasing i (suffixes)

B $x(|S|) = 0$

O $x(0)$

T $O(k^2|S|)$ work per the recurrence

$+ O(k|P|)$ work to construct hash set to enable $O(1)$ substring (i.e. $S[i:j]$) look-ups

(1)

S $x(s, k, n)$:= The min. total drop height to determine w/ certainty the breaking floor given k eggs, a building of n floors, starting at floor s

R $x(f, k, n) := \min \{$

$hf + \max \{ x(hf + hf+f, k, n-f)$

$x(hf - hf-f, k-1, f-1) \}$

for $1 \le f \le n \}$

# of remaining floors to check

current floor

# of eggs left

NEED HEIGHTS, not FLOORS!

(?) I think this formulation also works, but not 100% sure... roughly same idea as solution modulo indexing explicitly vs. implicitly.

I ~~increasing~~ k, increasing n

B $x(f, k, 0) = 0$

$x(f, 1, n) = \sum_{i=1}^{n} i$

O $x(0, k, n)$

T $O(n^2 k)$ subproblems $\times$ $O(n)$ work/subproblem $= O(n^3 k)$ overall

Problem 9-5. Building a Wall

The pigs in Porkland from Problem Session 2, have decided to build a stone wall along their southern border for protection against the menacing wolf. The wall will be one meter thick, n meters long, and at most k meters tall. The wall will be built from a large supply of identical **long stones**: each a $1 \times 1 \times 2$ meter rectangular prism. Long stones may be placed either vertically or horizontally in the wall. With much difficulty, a single long stone can be broken into two 1-meter **cube stones**, but the pigs prefer not using cube stones when possible.

The ground along the southern border of Porkland is uneven, but the pigs have leveled each square meter along the border to an integer meter elevation. Let a **border plan** be an $n \times k$ array `B` correspond to what the border looks like before a wall has been built. `B[j][i]` corresponds to the cubic meter whose top is at elevation $k - j$, located at meter i along the border. `B[j][i]` is `'.'` if that cubic meter is **empty** and must be covered by a stone, and `'#'` if that cubic meter is **dirt**, so should not be covered. `B` has the property that if `B[j][i]` is covered by dirt, so is every cubic meter `B[t][i]` beneath it (for $t \in \{j, \ldots, k-1\}$), where the top-most cubic meter `B[0][i]` in each column is initially empty. Below is an example `B` for $n = 10$ and $k = 5$.

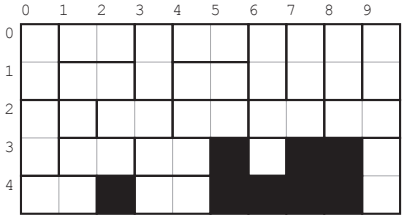
A **placement** of stones into border plan B is a set of placement triples:

- `(i,j,'1')` places a cube stone to cover `B[j][i]`;
- `(i,j,'D')` places a long stone oriented down to cover `B[j][i]` and `B[j + 1][i]`; and
- `(i,j,'R')` places a long stone oriented right to cover `B[j][i]` and `B[j][i + 1]`.

A placement is **complete** if every empty cubic meter in `B` is covered by some stone; and is **non-overlapping** if no cubic meter is covered by more than one stone and no stone overlaps dirt. Below is a complete non-overlapping placement for `B` that uses 2 cube stones, and a pictorial depiction.

```
B = [
    '..........',
    '..........',
    '..........',
    '.....#.##.',
    '..#..####.',
]
```

```
P = [
    (0,0,'D'), (0,2,'D'), (0,4,'R'), (1,0,'R'), (1,1,'R'),
    (1,2,'1'), (1,3,'R'), (2,2,'R'), (3,0,'D'), (3,3,'R'),
    (3,4,'R'), (4,0,'R'), (4,1,'R'), (4,2,'R'), (6,0,'D'),
    (6,2,'R'), (6,3,'1'), (7,0,'D'), (8,0,'D'), (8,2,'R'),
    (9,0,'D'), (9,3,'D'),
]
```



SKIPPED WRITE-UP FOR TIME; TALKED THROUGH W/ RCers.

(a) Given $n \times k$ border plan B, describe an $O(2^{2k}kn)$-time algorithm to return a complete non-overlapping placement for B using the fewest cube stones possible.

(b) Write a Python function `build_wall(B)` that implements your algorithm from (a) **for border plans with $k = 5$.**

MIT OpenCourseWare
https://ocw.mit.edu

6.006 Introduction to Algorithms

Spring 2020

For information about citing these materials or our Terms of Use, visit: https://ocw.mit.edu/terms