Ravi Dayabhai

Problem Set 7

Please write your solutions in the LaTeX and Python templates provided. Aim for concise solutions; convoluted and obtuse descriptions might receive low marks, even when they are correct.

Please solve each of the following problems using **dynamic programming**. For each problem, be sure to define a set of subproblems, relate the subproblems recursively, argue the relation is acyclic, provide base cases, construct a solution to the original problem from the subproblems, and analyze running time. Correct but inefficient dynamic programs will be awarded significant partial credit.

Problem 7-1. [15 points] **Effective Campaigning** (SEE CODE)

Representative Zena Torr is facing off against Senator Kong Grossman in a heated presidential primary: a sequence of n head-to-head state contests, one per day for n days. Each state contest $i \in \{1, \ldots, n\}$ has a known positive integer **delegate count** d_i, and a **projected delegate count** $z_i < d_i$ that Rep. Torr would win if she took no further action. There are $D = \sum_i d_i$ total delegates and Rep. Torr needs at least $\lfloor D/2 \rfloor + 1$ delegates to win. Unfortunately, Rep. Torr is projected to lose the race, since $\sum_i z_i < \lfloor D/2 \rfloor + 1$, so she needs to take action. Rep. Torr has a limited but effective election team which can **campaign** in at most one state per day. If the team campaigns on day i, they will win all d_i delegates in state i, but they will **not be able to campaign at all** for two days after day i, as it will take time to relocate. Describe an $O(n)$-time algorithm to determine whether it is possible for Rep. Torr to win the primary contest by campaigning effectively.

Problem 7-2. [15 points] **Caged Cats** (SEE NEXT PAGE)

Ting Kiger is an eccentric personality who owns n pet tigers and n^2 cages.

- Each tiger i has known positive integer **age** a_i and **size** s_i (no two have the same age or size).
- Each cage j has known positive integer **capacity** c_j and **distance** d_j from Ting's bedroom (no two have the same capacity or distance).

Ting needs to assign each tiger its own cage.

- Ting **favors older tigers** and wants them to sleep closer to his bedroom, i.e., any two tigers x and y with ages $a_x < a_y$ must be assigned to cages X and Y respectively such that $d_Y < d_X$.
- A tiger i assigned to cage c_j will experience positive **discomfort** $s_i - c_j$ if $s_i > c_j$, but will not experience any discomfort if $s_i \leq c_j$.

Describe an $O(n^3)$-time algorithm to assign tigers to cages that favors older tigers and minimizes the total discomfort experienced by the tigers.

②

S Let $\bigcirc^{i,j}$ be the minimum discomfort placing tigers $\{i, \ldots n\}$ among cages $\{j \ldots n^2\}$ where tigers sorted desc by age and cages asc. in distance

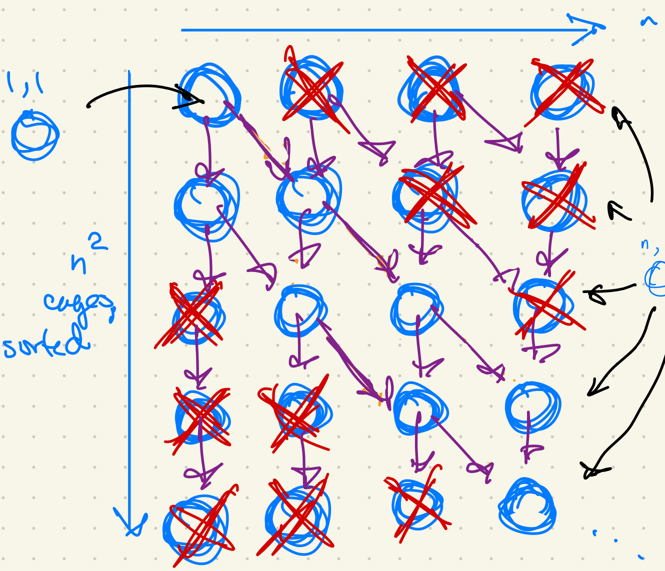
R $\bigcirc^{i,j} := \min \Big\{ \bigcirc^{i+1,\, j+1} +$ discomfort from putting i^{th} oldest tiger in j^{th} closest cage , $\bigcirc^{i,\, j+1} \Big\}$

$\cancel{\bigcirc}^{i,j} := \infty$ when $\begin{cases} j \le i-1 & \text{or} \\ j > n^2 - n + i \end{cases}$

T decreasing j, decreasing i

O

$\bigcirc^{1,1}$



B

$\bigcirc^{n,j} := \min\big($ discomfort putting n^{th} tiger in j^{th} cage , $\bigcirc^{n,\, j+1} \big)$

$\bigcirc^{n,n^2} :=$ discomfort putting n^{th} oldest tiger in $(n^2)^{th}$ closest cage.

T $O(n^3)$ subproblems $\times$ $O(1)$ work per subproblem

$\Rightarrow O(n^3)$ overall run time

(3)

$\boxed{S}$ $x(v) :=$ # of odd length paths from s to v

$y(v) :=$ # of even length paths from s to v

$\boxed{R}$
$$x(v) := \sum_{u \in Adj^-(v)} \begin{cases} \text{if } w(u,v) \text{ is even}, & x(u) \\ \text{if } w(u,v) \text{ is odd}, & y(u) \end{cases}$$

$$y(v) := \sum_{u \in Adj^-(v)} \begin{cases} \text{if } w(u,v) \text{ is even}, & y(u) \\ \text{if } w(u,v) \text{ is odd}, & x(u) \end{cases}$$

$\boxed{T}$ topological ordering on G, starting from s

$\boxed{B}$ $x(s) := 0$

$y(s) = 1$ (zero edges/edgeweight path is even)

$\begin{cases} x(v) \\ y(v) \end{cases} = 0$ when $Adj^-(v) = \emptyset$,

$\boxed{O}$ $x(t)$

$\boxed{T}$ $O(|V|+|E|)$ time = (think: traversing DAG and doing $O(1)$ work at each node)

$O(|V|)$ subproblems ×

$O(\deg^-(v))$ work per subproblem

(SEE PREV. PAGE)

Problem 7-3. [15 points] **Odd Paths**

Given a weighted directed acyclic graph $G = (V, E, w)$ with integer weights and two vertices $s, t \in V$, describe a linear-time algorithm to determine the number of paths from s to t having **odd weight**. When solving this problem, you may assume that a single machine word is large enough to hold any integer computed during your algorithm.

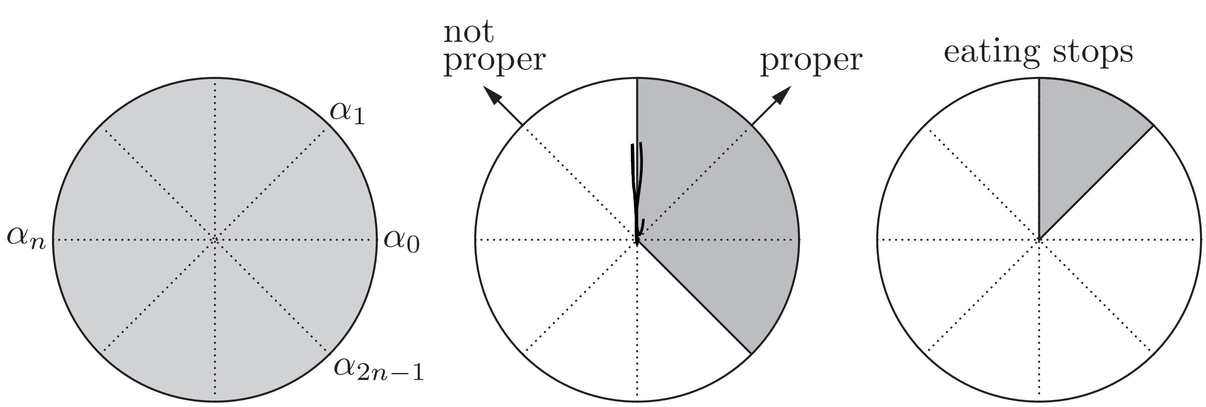
Problem 7-4. [15 points] **Pizza Partitioning**

Almost right idea...

Liza Pover and her little brother Lie Pover want to share a round pizza pie that has been cut into $2n$ equal sector slices along rays from the center at angles $\alpha_i = i\pi/n$ for $i \in \{0, 1, \ldots, 2n\}$, where $\alpha_0 = \alpha_{2n}$. Each slice i between angles α_i and α_{i+1} has a known integer tastiness t_i (which might be negative). To be "fair" to her little brother, Liza decides to eat slices in the following way:

- They will each take turns choosing slices of pizza to eat: Liza starts as **the chooser**.
- If there is only one slice remaining, the chooser eats that slice, and eating stops.
- Otherwise the chooser does the following:
  - Angle α_i is **proper** if there is at least one uneaten slice on either side of the line passing through the center of the pizza at angle α_i.
  - The chooser picks any number $i \in \{1, \ldots, 2n\}$ where α_i is proper, and eats all uneaten slices counter-clockwise around the pizza from angle α_i to angle $\alpha_i + \pi$.
  - Once the chooser has eaten, the other sibling becomes the chooser, and eating continues.

Liza wants to maximize the total tastiness of slices she will eat. Describe an $O(n^3)$-time algorithm to find the maximum total tastiness Liza can guarantee herself via this selection process.



4

$\boxed{S}$ Let $x(i)(j)$ be the maximum tastiness possible when slices from $a_{i \bmod 2n}$ to $a_{j \bmod 2n}$ are left uneaten.

$\boxed{R}$ $x(i)(j) := \max \{$

$$\sum_{l=i+k}^{\min(j-i-1,\, i+n)} t_{l \bmod 2n} + \left(\sum_{m=i}^{k} t_{m \bmod 2n} - x(i)(k) \right)$$

where $k \in \{1, \ldots, j-i-1 \bmod 2n\}$

$\boxed{T}$ increasing $j-i \bmod 2n$

$\boxed{B}$ $x(i)(i+1) = t_i$

$\boxed{O}$ $x(0)(2n)$

$\boxed{T}$ There are $\Theta(n^2)$ "substrings" of slices where each "substring" is given by $i, j \in \{0, \ldots, 2n\}$. $\Theta(n)$ non-recursive work is done in each $x(i)(j)$ call $\Rightarrow \Theta(n^3)$ run time.

Given solution uses subproblem expansion ("min-max"), but this has elements of correct relation — formalization was tough!

Problem 7-5. [40 points] **Shorting Stocks**

Bordan Jelfort is a short seller at a financial trading firm. He has collected **stock price information** from s different companies $C = (c_0, \dots, c_{s-1})$ for n consecutive days. Stock price information for a company c_i is a chronological sequence $P_i = (p_0, \dots, p_{nk-1})$ of nk **prices**, where each price is a positive integer and prices $\{p_{kj}, \dots, p_{kj+k-1}\}$ all occur on day j for $j \in \{0, \dots, n-1\}$. The **shorting value** of a company is the length of the longest chronological subsequence of strictly decreasing prices for that company that **doesn't skip days**: if the sequence contains two prices on different days i and j with $i < j$, then the sequence must also contain at least one price from every day in $\{i, \dots, j\}$.

(a) [15 points] Describe an $O(snk^2)$-time algorithm to determine which company c_i has the highest shorting value, and return a longest subsequence S of decreasing subsequences of prices from P_i that doesn't skip days.

(b) [25 points] Write a Python function `short_company(C, P, n, k)` that implements your algorithm from part (a) using the template code provided. You can download the code template and some test cases from the website. (SEE CODE)

```python
def short_company(C, P, n, k):
    '''
    Input:  C | Tuple of s = |C| strings representing names of companies
            P | Tuple of s lists each of size nk representing prices
            n | Number of days of price information
            k | Number of prices in one day
    Output: c | Name of a company with highest shorting value
            S | List containing a longest subsequence of
              | decreasing prices from c that doesn't skip days
    '''
    c = C[0]
    S = []
    ##################
    # YOUR CODE HERE #
    ##################
    return (c, S)
```