Problem Set 7

All parts are due on April 26, 2020 at 6PM. 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. Solutions should be submitted on the course website, and any code should be submitted for automated checking on `alg.mit.edu`.

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**

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.

Solution:

1. **Subproblems**
   - $x(i)$: max delegates Torr can win in states i to n, where campaign on day i is possible
   - for $i \in \{1, \ldots, n+1\}$
2. **Relate**
   - Will either campaign or not on day i. Guess!
   - If campaign, get d_i delegates, but cannot campaign during next two days
   - Otherwise, no restriction, but only get z_i delegates
   - $x(i) = \max\{d_i + z_{i+1} + z_{i+2} + x(i+3), z_i + x(i+1)\}$

3. **Topo**
   - $x(i)$ only depends on subproblems with strictly larger i, so acyclic
4. **Base**
   - $x(n+1) = 0$ (no more states to be won)
   - $x(n) = d_n$ (campaign on last day, since $d_n > z_n$)
   - $x(n-1) = \max\{d_{n-1} + z_n, z_{n-1} + d_n\}$ (campaign on either of last two days)
5. **Original**
   - Solve subproblems via recursive top down or iterative bottom up
   - Return whether $x(1) \geq \lfloor D/2 \rfloor + 1$ (the maximum delegates she can achieve is at least as large as the amount she needs to win)
6. **Time**
   - # subproblems: $n + 1$
   - Work per subproblem: $O(1)$
   - $O(n)$ running time

Rubric: (Same for all theory problems in PS7)

- S: 3 points for a correct subproblem description
- R: 3 points for a correct recursive relation
- T: 1 points for indication that relation is acyclic
- B: 1 point for correct base cases in terms of subproblems
- O: 1 point for correct original solution in terms of subproblems
- T: 2 points for correct analysis of running time
- 4 points if a correct dynamic program is efficient (meets requested bound)
- Partial credit may be awarded

Problem 7-2. [15 points] **Caged Cats**

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.

Solution: The "favors older tigers" condition ensures that the tigers sorted decreasing by age will be matched with a subsequence of the cages sorted increasing by distance from Ting's bedroom. So first sort the tigers into sequence T decreasing by age in $O(n \log n)$ time and the cages into sequence C increasing by distance in $O(n^2 \log n)$ time (e.g., via merge sort). Now, we use dynamic programming to find an optimal matching from T with a subsequence of C.

1. **Subproblems**
   - $x(i, j)$: min total discomfort possible, matching tigers $T[i:]$ to a subsequence of $C[j:]$
   - for $i \in \{0, \ldots, n\}$ and $j \in \{0, \ldots, n^2\}$
2. **Relate**
   - Can either match tiger $T[i]$ with cage $C[j]$ or not. Guess!
   - If matched, incur discomfort of match (if any).
   - In either case, won't match $C[j]$ with any tiger in $T[i+1:]$ so recurse on remainder
   - Let $d(i, j)$ be the discomfort of matching tiger $T[i]$ with cage $C[j]$
   - i.e., $d(i, j) = s_i - c_j$ if $s_i > c_j$ and zero otherwise
   - $x(i, j) = \min\{d(i, j) + x(i+1, j+1), x(i, j+1)\}$
3. **Topo**
   - $x(i, j)$ only depends on subproblems with strictly larger j, so acyclic
4. **Base**
   - $x(n, j) = 0$ (all tigers are matched, so no discomfort)
   - $x(i, n^2) = \infty$ for $i > 0$ (no more cages and cannot have homeless tigers)
5. **Original**
   - Solve subproblems via recursive top down or iterative bottom up
   - $x(0, 0)$ is the minimum total discomfort matching all tigers to cages
   - Store parent pointers to reconstruct the optimal assignment
6. **Time**
   - # subproblems: $(n+1)(n^2+1) = O(n^3)$
   - Work per subproblem: $O(1)$
   - $O(n^3)$ running time

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.

Solution: The number of paths from s to t having odd weight, depends on the number of paths to each of t's incoming neighbors: paths of odd weight if the incoming edge is even, and paths of even weight if the incoming edge is odd. So we make two types of subproblem per vertex representing the number of even weight and odd weight paths to the vertex.

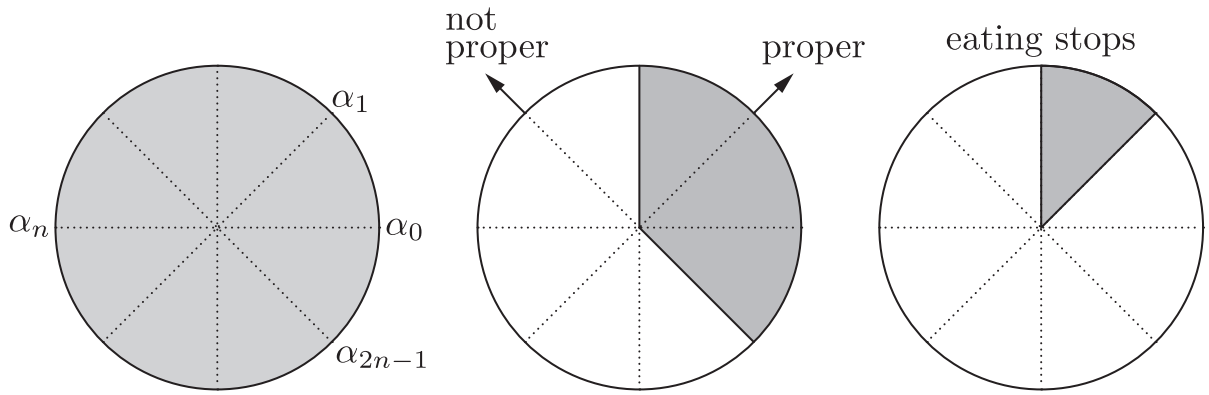
1. **Subproblems**
   - $x(v, i)$: the number of paths from s to v, having even weight if $i = 0$ and odd weight if $i = 1$
   - for all $v \in V$ and $i \in \{0, 1\}$
2. **Relate**
   - In a DAG, the number of even or odd paths to v is the sum of the relevant paths to its incoming vertices
   - Let $p(i)$ be the **parity** of integer i, i.e., $p(i) = 0$ if i is even and $p(i) = 1$ if i is odd
   - $x(v, i) = \sum\{x(u, p(w(u, v) + i)) \mid u \in \text{Adj}^-(v)\}$
3. **Topo**
   - $x(v, i)$ depends only on subproblems $x(u, j)$ for vertices u that appear earlier in the topological order of G, so acyclic
4. **Base**
   - $x(s, 0) = 1$ (since zero edges is even)
   - $x(s, 1) = 0$ (no odd length paths to s from s)
   - $x(v, 0) = x(v, 1) = 0$ for any other v where $\text{Adj}^-(v) = \emptyset$
5. **Original**
   - $x(t, 1)$, the number of odd paths from s to vertex t
6. **Time**
   - # subproblems: $2|V|$
   - Work per subproblem: $O(\deg^-(v))$
   - $O(2\sum_{v \in V} \deg^-(v)) = O(|V| + |E|)$ running time (linear in the size of the graph)

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

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.



Solution: As the siblings eat pizza, the uneaten slices are always cyclically consecutive (since removal of a half plane is a convex operation). So we choose consecutive cyclic subarrays as our subproblems. Each sibling wants to maximize tastiness, we make different subproblems based on which sibling is currently the chooser.

While making choices, it will be useful to know the tastiness of any subarray. Let $v(i, j)$ be the tastiness of the j slices counter-clockwise from angle α_i, i.e., $v(i, j) = \sum_{k=0}^{j-1} t_{((i+k) \bmod 2n)}$, where indices are taken modulo $2n$, for $i \in \{0, \ldots, 2n-1\}$ and $j \in \{0, \ldots, n\}$. There are $O(n^2)$ such $v(i, j)$ and each can be computed naïvely in $O(n)$ time, $O(n^3)$ time in total. Note that these can also be computed in $O(n^2)$ time via dynamic programming, but the naïve method is sufficient for the requested time bound.

1. **Subproblems**
   - $x(i, j, p)$: the maximum tastiness Liza can acheive with the j slices counter-clockwise from α_i remaining, when either: Liza is the chooser when $p = 1$, or Lie is the chooser when $p = 2$
   - for $i \in \{0, \ldots, 2n-1\}$, $j \in \{0, \ldots, n\}$, $p \in \{1, 2\}$
   - (we can only have at most n slices after first choice, so we don't compute for $j > n$)
2. **Relate**
   - Liza tries to maximize, while Lie may try to minimize (so in worst case he does)
   - Chooser can choose any proper angle and then choose a side. Guess!
   - Angle α_{i+k} is proper for any $k \in \{1, \ldots, j-1\}$
   - Chooser eats either: k slices between α_i and α_{i+k} or $j-k$ slices between α_{i+k} and α_{i+j}
   - Liza does not gain or lose tastiness when Lie eats
   - $x(i, j, 1) = \max\left\{ \max\left\{ \begin{array}{l} v(i,k) + x(i+k, j-k, 2), \\ v(i+k, j-k) + x(i, k, 2) \end{array} \right\} \middle| \, k \in \{1, \ldots, j-1\} \right\}$
   - $x(i, j, 2) = \min\left\{ \min\left\{ \begin{array}{l} x(i+k, j-k, 1), \\ x(i, k, 1) \end{array} \right\} \middle| \, k \in \{1, \ldots, j-1\} \right\}$
3. **Topo**
   - $x(i, j, p)$ only depends on subproblems with strictly smaller j, so acyclic
4. **Base**
   - $x(i, 1, 1) = t_i$ (Liza eats last slice)
   - $x(i, 1, 2) = 0$ (Lie eats last slice)
   - for all $i \in \{1, \ldots, 2n\}$
5. **Original**
   - Liza first chooses which half to eat
   - Max is sum of tastiness on a half and tastiness achieved letting Lie choose on other half
   - $\max\{x(i, n, 2) + v(((i+n) \bmod 2n), n) \mid i \in \{0, \ldots, 2n-1\}\}$
6. **Time**
   - Work for compute all $v(i, j)$: $O(n^3)$
   - # subproblems: $2(2n)(n+1) = O(n^2)$
   - Work per subproblem: $O(n)$
   - Work to compute original: $O(n)$
   - $O(n^3)$ running time

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 $C = (c_0, \ldots, c_{s-1})$ for n consecutive days. Stock price information for a company c_i is a chronological sequence $P_i = (p_0, \ldots, p_{nk-1})$ of nk **prices**, where each price is a positive integer and prices $\{p_{kj}, \ldots, p_{kj+k-1}\}$ all occur on day j for $j \in \{0, \ldots, n-1\}$. 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 $\{i, \ldots, j\}$.

(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.

Solution: We will compute the shorting value of each company via dynamic programming (assuming $P = P_i$), and then return the longest in $O(s)$ time.

1. **Subproblems**
   - $x(j)$: the longest decreasing subsequence of prices that doesn't skip days in $P[j:]$ that includes price $P[j]$
   - for $j \in \{0, \ldots, nk-1\}$
2. **Relate**
   - Next in sequence has price at later time in same day or the next day. Guess!
   - At price index j, there are $(k-1) - (j \bmod k)$ prices left in same day
   - Then there are k prices in the following day (if it exists)
   - So last next price index is $f(j) = \min\{j + (k-1) - (j \bmod k) + k, nk-1\}$
   - $x(j) = 1 + \max\{x(d) \mid d \in \{j+1, \ldots, f(j)\} \text{ and } P[j] > P[d]\} \cup \{0\}$
3. **Topo**
   - $x(j)$ only depends on subproblems with strictly larger j, so acyclic
4. **Base**
   - Relation includes base case when minimization contains only 0
   - Can also say: $x(nk-1) = 1$ (only one item left)
5. **Original**
   - Solve subproblems via recursive top down or iterative bottom up
   - Shorting value is $\max_{j=0}^{nk-1} x(j)$
   - Store parent pointers to reconstruct S
6. **Time**
   - # subproblems: nk
   - Work per subproblem: $O(k)$
   - Work for original: $O(nk)$
   - $O(nk^2)$ running time

(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. Submit your code online at `alg.mit.edu`.

Solution:

```
# iterative bottom-up
def short_company(C, P, n, k):
    S = []
    for i in range(len(C)):
        p = P[i]
        x = [1    for _ in range(n*k)]       # subproblem memo
        r = [None for _ in range(n*k)]       # parent pointer memo
        best = 0
        for j in range(n*k - 1, -1, -1):     # compute memos
            f = min(j + (k - 1) - (j % k) + k, n*k - 1)
            for d in range(j + 1, f + 1):
                if (p[j] > p[d]) and (1 + x[d] > x[j]):
                    x[j] = 1 + x[d]
                    r[j] = d
                    if x[best] < x[j]:
                        best = j
        if x[best] > len(S):                 # reconstruct from parent pointers
            c, S = C[i], []
            while best != None:
                S.append(p[best])
                best = r[best]
    S = tuple(S)
    return (c, S)
```

```python
# recursive top-down
def short_company(C, P, n, k):
    S = []
    for i in range(len(C)):
        p = P[i]
        memo = [None for _ in range(n*k)]    # memo for subproblems and parents
        def dp(j):                           # recursive function
            if memo[j] is None:
                f = min(j + (k - 1) - (j % k) + k, n*k - 1)
                x, r = 1, None
                for d in range(j + 1, f + 1):
                    x_, _ = dp(d)
                    if (p[j] > p[d]) and (1 + x_ > x):
                        x, r = 1 + x_, d
                memo[j] = (x, r)
            return memo[j]
        best, opt = 0, 0
        for j in range(n*k):                 # compute subproblems
            x, _ = dp(j)
            if x > opt:
                best, opt = j, x
        if opt > len(S):                     # reconstruct from parent pointers
            c, S = C[i], []
            while best != None:
                S.append(p[best])
                _, best = dp(best)
    S = tuple(S)
    return (c, S)
```