PS 7

7-1.

For state i. { if Torr does nothing, get z_i
if Torr's election team campaigns, get d_i

If compaigns on day i, cannot compaign on day (i + 1). So the number of delegates Torr can win on day i, i + 1, i + 2 is $d_i + z_{i+1} + z_{i+2}$ if the team campaigns on day i.

Subproblems: Choosing prefixes.
x(i) ::= "The maximum delegates to win until day i by campaigning effectively."

Relate:
- Cases which are should be considered : campaigned on day i - 2, campaigned on day i - 1, didn't campaign both on day i - 2 and i - 1.
- $x(i) = \max\{x(i-1) + z_i, x(i-3) + z_{i-2} + z_{i-1} + d_i\}$

Correctness: x(i) indicates the maximum delegates to win until day i.
Pf. By induction on i. Assume that for x(i), true. Then check if for x(i + 1) also true.
$x(i+1) = \max\{ x(i) + z_{i+1}, x(i-2) + z_{i-1} + z_i + d_{i+1}\}$.
By induction, x(i) and x(i - 2) are maximum value for i and i - 2 respectively.
If for x(i) it is optimal to campaign on day i but for x(i + 1) it is optimal to campaign on day i + 1, then $x(i+1) = x(i-2) + z_{i-1} + z_i + d_{i+1}$
Otherwise, $x(i+1) = x(i) + z_{i+1}$ and this case includes the cases which campaigned on day i - 1, campaigned on day i. So x(i + 1) is optimal. □

Topo. order: increasing i for $i \in \{1, ..., n\}$.

Base case: x(0) = 0, $x(1) = d_1$, x(2) = max{...}

Original problem: True if $x(n) \geq \lfloor D/2 \rfloor + 1$, otherwise false

Time: O(n) subproblems and O(1)-time per a nonrecursive problem. So this algorithm takes O(n)-time.

7-2.

A tiger [a_i age ; s_i size] A cage [capacity c_j ; distance d_j] n tigers and n^2 cages

For any two tigers x, y, $a_x < a_y$ IMPLIES $d_y < d_x$

discomfort [$s_i - c_j$ if $s_i > c_j$; 0 otherwise]

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

Subproblems:
- The order of tigers are maintained.
- For i, j $\in$ Integer, i < j IFF $a_i < a_j$
- u(i, n) ::= "The minimum sum of discomfort experienced by n tigers assigned to i cages where $d_i < d_j$, keeping the condition that favors older tigers" for i = n, ... , n^2.

Relate:
- Assume u(k, n) where k >= n.
- For (k + 1) edges, the oldest tiger move to (k + 1)st cage.
- If the oldest tiger move to (k + 1)st cage, the remainder of tigers (n - 1 tigers) chooses their cages within k cages.
- So, $u(i, n) = \begin{cases} \min\{u(i-1, n),\ u(i-1, n-1) + s_n - c_i\}, & \text{if } s_n > c_i \\ \min\{u(i-1, n),\ u(i-1, n-1)\}, & \text{if } s_n \le c_i \end{cases}$

- But for u(i + 1, n), u(i, n - 1) should be defined, and for u(i, n - 1), u(i - 1, n - 2) needs to be defined. and so on.
- So, $u(i, k) = \begin{cases} \min\{u(i-1, k),\ u(i-1, k-1) + s_k - c_i\}, & \text{if } s_k > c_i \\ \min\{u(i-1, k),\ u(i-1, k-1)\}, & \text{if } s_k \le c_i \end{cases}$ for $k \in \{1, \dots, n\}$

- For some j, l where j <= i and l <= k, if it is optimal that l-th tiger is not in j-th cage, u(j, l) = u(j - 1, l).
If it is optimal that l-th tiger is in j-th cage, $u(j, l) = u(j - 1, l - 1) + \begin{cases} s_l - c_j & \text{if } s_l > c_j \\ 0 & \text{if } s_l \le c_j \end{cases}$
So the formula hold.

Topological order: increasing i.

Base case: for i $\in$ { 1, ... , n } and j $\in$ { 1, ... , i }, u(i, j).

Original problem: $u(n^2, n)$. Store parent pointers to reconstruct original problem.

Time: $O(n^2)$ subproblems. For a subproblem, u(i, 1), ..., u(i, n) must be defined, so O(n)-time per a subproblem. Thus, $O(n^2) * O(n) = O(n^3)$-time.

7-3.
Integer weight. DAG.
A path should have the odd number of edges having odd weight.
First of all, get a topological order T by full dfs.
For every u, v $\in$ V, T(u) < T(v) IFF u is followed by v in T.

If T(s) > T(v), abort and the number of paths from s to t having odd weight is 0.

Subproblems:
- $p_o(x)$ ::= "The number of paths from s to x having odd weight"
  $p_e(x)$ ::= "The number of paths from s to x having even weight"

Relate:
- $p_o(x) = \sum_{u \in Adj^-(v)} \begin{bmatrix} p_e(u), & \text{if } w(u,v) \text{ is odd} \\ p_o(u), & \text{if } w(u,v) \text{ is even.} \end{bmatrix}$

$p_e(x) = \sum_{u \in Adj^-(v)} \begin{bmatrix} p_o(u), & \text{if } w(u,v) \text{ is odd} \\ p_e(u), & \text{if } w(u,v) \text{ is even} \end{bmatrix}$ for some $u, v \in V$ where $u \in Adj^-(v)$

- By the definition of topological order, $u \in Adj^-(v)$ IMPLIES $T(u) < T(v)$
- Since DAG Relaxation is safe, for $u \in Adj^-(v)$, $p_o(u)$ and $p_e(u)$ were determined.

Topological order: G is a dag.

Base case: $p_o(s) = 0$, $p_e(s) = 0$

Original Problem: $p_o(t)$

Time: Getting T takes O(|V| + |E|)-time by full dfs.
$O(\sum_{v \in V} deg^-(v)) = O(|E|)$.
So the algorithm takes O(|V| + |E|)-time.

7-4.
Subproblems:
- After Liza makes her first choice, there remain n slices of pizza consisting of a half circle.
- For the half circle, it is not possible to eat both sides of slices by one choice. Thus, the remained n slices of pizza consisting of a half circle as a sequence of slices.
- If Liza chooses α_i for the first time, remainders of slices are
  i + n, i + n +1, ... , i + 2n - 1 = i - 1.
- x(a, b, c) ::= "For chooser $c \in$ { Liza, Lie }, the maximum tastiness able to get from slices ranging from a to b."

Relate:
- $x(a, b, Liza) = \max\{ x(a + k, b, Lie) + \sum_{i=a}^{a+k-1} t_i,\ x(a, b - k, Lie) + \sum_{i=b-k+1}^{b} t_i \mid k \in \{ 1, \dots , b - a \} \}$
  $x(a, b, Lie) = \max\{ x(a + k, b, Liza), x(a, b - k, Liza) \mid k \in \{ 1, \dots , b - a \} \}$.

Topological order: decreasing b - a

Base case: $x(i, i, Liza) = t_i$, $x(i, i, Lie) = 0$ for i = 0, ... , 2n - 1

Original problem: $\max\{ x(i, i + n - 1, \text{Lie}) + \sum_{i \le m}^{i+2n-1} t_i \mid i \in \{1, \ldots, 2n-1\} \}$

Time:
O(n^2) subproblems
O(1)-time for a nonrec. problem.
O(n)-iteration
So O(n^3)-time.