Introduction to Algorithms **Solution Set 3**
CS 4820, Spring 2019

(1) *(10 points) In American politics,* gerrymandering *refers to the process of subdividing a region into electoral districts to maximize a particular political party's advantage. This problem explores a simplified model of gerrymandering in which the region is modeled as one-dimensional. Assume that there are* $n > 0$ precincts *represented by the vertices* $v_1, v_2, \ldots, v_n$ *of an undirected path. A* district *is defined to be a contiguous interval of precincts; in other words a district is specified by its endpoints* $i \leq j$*, and it consists of precincts* $v_i, v_{i+1}, v_{i+2}, \ldots, v_j$*. We will refer to such a district as* $[i, j]$*.*

Assume there are two parties A and B competing in the election, and for every $1 \leq i \leq j \leq n$*,* $P[i, j]$ *denotes the probability that A wins in the district* $[i, j]$*. The probability matrix P is given as part of the input. Assume that the law requires the precincts to be partitioned into exactly k disjoint districts, each containing at least* $s_{\min}$ *and at most* $s_{\max}$ *nodes. You may also assume the parameters* $n, k, s_{\min}, s_{\max}$ *are chosen such that there is at least one way to partition the precincts into k districts meeting the specified size constraints. Your task is to find an efficient algorithm to gerrymander the precincts into k districts satisfying the size constraints, so as to maximize the expected number of districts that A wins.*

Solution. Define $OPT(r, \ell)$ to be the optimal expected value of the number of districts A wins in an election involving precincts $1, \ldots, r$ and gerrymandering them into exactly ℓ districts. To ensure that $OPT(r, \ell)$ is well-defined in all cases: if there is no way to partition $1, \ldots, r$ into exactly ℓ districts of sizes between $s_{\min}$ and $s_{\max}$ then define $OPT(r, \ell)$ to be $-\infty$. Our problem is to compute $OPT(n, k)$.

Computing $OPT(r, \ell)$: if $\ell = 1$ then there is at most one way to split $1, \ldots, r$ into ℓ districts satisfying the specified size bounds. Thus, $OPT(r, \ell)$ equals $P[1, r]$ for $s_{\min} \leq r \leq s_{\max}$ and $-\infty$ otherwise. If $\ell > 1$ then in the optimal partition the district that contains precinct r is of the form $[j, r]$ and the remaining districts partition $1, \ldots, j-1$ into $\ell - 1$ pieces. Using linearity of expectation, it follows that $OPT(r, \ell) = OPT(j-1, \ell-1) + P[j, r]$. Since there are bounds of $s_{\min}$ and $s_{\max}$ on the size of a district, it implies that $r - s_{\max} + 1 \leq j \leq r - s_{\min} + 1$. Let $I(r)$ denote the set of all $j \geq 1$ satisfying $r - s_{\max} + 1 \leq j \leq r - s_{\min} + 1$. From the discussion, we get the following recursive relation.

$$OPT(r, \ell) = \begin{cases} P[1, r] & \text{if } \ell = 1 \text{ and } s_{\min} \leq r \leq s_{\max} \\ -\infty & \text{if } \ell = 1 \text{ and } r < s_{\min} \text{ or } r > s_{\max} \\ \max\{OPT(j-1, \ell-1) + P[j, r] : j \in I(r)\} & \text{if } \ell > 1 \end{cases} \quad (1)$$

It is now straightforward to develop a dynamic programming solution based on the above recursion.

1. for $r = 1$ to n
2. for $\ell = 1$ to k
3. let $T[r, \ell] = \begin{cases} P[1, r] & \text{if } \ell = 1 \text{ and } s_{\min} \leq r \leq s_{\max} \\ -\infty & \text{if } \ell = 1 \text{ and } r < s_{\min} \text{ or } r > s_{\max} \\ \max\{T[j-1, \ell-1] + P[j, r] : j \in I(r)\} & \text{if } \ell > 1 \end{cases}$
4. Output $T(n, k)$

The correctness of the above procedure relies on an inductive argument as is usual in dynamic programming algorithms. In the inductive argument, we will assume that $OPT(r, \ell)$ satisfies the recurrence relation (1), since the validity of that equation was proven in the paragraph preceding the equation. We use the following inductive hypothesis parameterized by ℓ:

$$\textit{For all } r \in \{1, \ldots, n\},\ T[r, \ell] = OPT(r, \ell).$$

The base case is when $\ell = 1$; the formula for $T[r, 1]$ given in the algorithm exactly matches the formula for $OPT(r, 1)$ given in equation (1). For the inductive step, assume the induction hypothesis is true for $\ell - 1$. Then for every r in $\{1, \ldots, n\}$ the value of $T[r, \ell]$ computed by the algorithm satisfies

$$\begin{aligned} T[r, \ell] &= \max\{T[j-1, \ell-1] + P[j, r] : \max\{r - s_{\max} + 1, 1\} \le j \le r - s_{\min} + 1\} \\ &= \max\{OPT(j-1, \ell-1) + P[j, r] : \max\{r - s_{\max} + 1, 1\} \le j \le r - s_{\min} + 1\} \\ &= OPT(r, \ell) \end{aligned}$$

where the first line comes from the pseudocode for the algorithm, the second line is derived by applying the induction hypothesis to each of the terms $T[j-1, \ell-1]$ appearing in the formula, and the third line is a consequence of equation (1). The chain of equations establishes the relation $T[r, \ell] = O(r, \ell)$ which completes the induction step and finishes the proof of correctness.

The running time of this algorithm is $O(nk(s_{\max} - s_{\min} + 1))$ because the inner and outer loops iterate n times and k times, respectively, leading to $O(nk)$ iterations of the inner loop in total. Each inner loop iteration requires computing the right-hand side of the recursive relation above, which is a maximum of at most $s_{\max} - s_{\min} + 1$ terms. Each term in the maximum can be computed using a constant number of arithmetic operations. Hence, there are $O(nk)$ loop iterations each requiring $O(s_{\max} - s_{\min} + 1)$ time, leading to the time bound of $O(nk(s_{\max} - s_{\min} + 1))$ as stated above.

(2) *(15 points) In the sport of American football, teams move a ball toward the opposing team's goal line in a sequence of plays called an offensive drive. Subject to some simplifications, the game has the following rules.*

1. *An offensive drive starts n yards away from the other team's goal line. In American football, n is often but not always equal to 80.*
2. *To retain possession of the ball, the team must move forward at least k yards in its first d plays, called "downs". In American football, $k = 10$ and $d = 4$.*
3. *More generally, certain plays in an offensive drive are called "first downs". The initial play in the drive is a first down. A subsequent play is called a first down if and only if the total yardage accumulated since the preceding first down is greater than or equal to k.*
4. *There are three ways that a drive could end.*
   - *If the total yardage accumulated in the drive is greater than or equal to n, the team scores a touchdown.*
   - *If the team has completed d plays since the last first down, and the total yardage accumulated in those d plays is less than k, the team loses possession.*
   - *Any play could result in a "turnover". If this happens, the team loses possession immediately.*
5. *On any given play of the drive, the team needs to decide whether to try a passing play or a running play. The probability of a turnover, and the distribution of the random number of yards gained in the event that the play is not a turnover, depend on whether the team chooses passing or running. We will assume that the number of yards gained is always a non-negative integer.*

A football strategy *is a strategy for choosing between a running or passing play in every possible situation that may arise during an offensive drive. A strategy is* optimal *if it maximizes the probability of scoring a touchdown.*

For an integer y in the range $0 \leq y \leq n$, let p_y denote the probability of gaining y yards on a passing play, and let r_y denote the probability of gaining y yards on a running play. Let p_{n+1} and r_{n+1} denote the probability of a turnover on a passing or running play, respectively. Assume these are non-negative rational numbers and that $\sum_{y=0}^{n+1} p_y = \sum_{y=0}^{n+1} r_y = 1$. Design an algorithm which is given the parameters n, k, d and the probabilities p_y, r_y for $y = 0, 1, \ldots, n+1$, and which determines whether the optimal football strategy chooses to run or to pass at the start of an offensive drive, n yards away from the opponent's goal line. Your algorithm should work for any *values of n, k, d, not only for $n = 80, k = 10, d = 4$.*

Solution. The state of play during the offensive drive can be represented by an ordered triple of positive integers (r, s, t) indicating that the team is r yards away from their opponent's goal line, must gain s yards to reach a first down, and has t downs remaining before losing possession. The rules of the game can be summarized by a state transition function $\textsc{Advance}(y, r, s, t)$ that outputs the state of play that is reached if the team starts in state (r, s, t) and runs a play that gains y yards. The output of the function $\textsc{Advance}(y, r, s, t)$ is either an ordered triple (r', s', t') or one of the special values $\{\top, \bot\}$ where $\top$ represents scoring a touchdown and $\bot$ represents losing possession. The rules of the game as described in the problem imply the following procedure for computing $\textsc{Advance}(y, r, s, t)$.

```
ADVANCE(y, r, s, t) :
      // Compute the result of gaining y yards starting from state (r, s, t)
if y ≥ r then                 // Touchdown
  | return ⊤
else if y ≥ s then            // First down
  | r' ← r − y
  | s' ← k
  | t' ← d
else if t = 1 then            // d downs elapsed without advancing k yards
  | return ⊥
else
  | r' ← r − y
  | s' ← s − y
  | t' ← t − 1
end
return (r', s', t')
```

The helper function $\textsc{Advance}(y, r, s, t)$ will greatly simplify the process of explaining the dynamic programming algorithm to compute an optimal football strategy.

Let $v^*(r, s, t)$ denote the probability of scoring a touchdown if the team plays the optimal strategy starting from state (r, s, t). Let $v^P(r, s, t)$ denote the probability of scoring a touchdown if the team does a passing play starting from state (r, s, t) followed by the optimal strategy on every subsequent play. Define $v^R(r, s, t)$ similarly, but starting with a running play rather than a passing play. Then,

letting $v^*(\top) = 1$ and $v^*(\bot) = 0$ for convenience, we have

$$v^P(r,s,t) = \sum_{y=0}^{n} p_y \cdot v^*(\textsc{Advance}(y,r,s,t)) \tag{2}$$

$$v^R(r,s,t) = \sum_{y=0}^{n} r_y \cdot v^*(\textsc{Advance}(y,r,s,t)) \tag{3}$$

$$v^*(r,s,t) = \max\{v^P(r,s,t),\, v^R(r,s,t)\}. \tag{4}$$

The first two lines simply express the summation, over all possible outcomes of the passing play (respectively, running play) of the probability of that outcome multiplied by the conditional probability of scoring a touchdown given that outcome, assuming optimal play for the remainder of the offensive drive. (The summations don't explicitly account for the probability of a turnover, but turnovers contribute zero to the probability of scoring a touchdown so there is no need to explicitly include that outcome in the sum.) Equation (4) asserts that the optimal strategy in state (r,s,t) starts with a passing or running play according to whether $v^P(r,s,t)$ or $v^R(r,s,t)$ is larger, and plays optimally thereafter. Intuitively this is obvious; rigorously, the way to prove it is as follows. First of all, $v^*(r,s,t) \geq \max\{v^P(r,s,t), v^R(r,s,t)\}$ because two of the potential strategies in state (r,s,t) are "passing followed by optimal play" and "running followed by optimal play", so the optimal strategy is at least as good as the better of these two alternatives. To prove the reverse inequality, suppose π is any strategy that proposes a passing play in state (r,s,t). Then letting $v^\pi(r',s',t')$ denote the probability of scoring a touchdown when using strategy π starting from any state (r',s',t'), we have

$$v^\pi(r,s,t) = \sum_{y=0}^{n} p_y \cdot v^\pi(\textsc{Advance}(y,r,s,t)) \leq \sum_{y=0}^{n} p_y \cdot v^*(\textsc{Advance}(y,r,s,t)) = v^P(r,s,t).$$

The first equation expresses that the unconditional probability of scoring a touchdown using π is the sum, over all outcomes of the initial passing play, of the probability of that outcome times that conditional probability of touchdown given that outcome. The last equation on the line is the definition of $v^P(r,s,t)$. The inequality in the middle of the line comes from the definition of v^*. An analogous calculation justifies the inequality $v^\pi(r,s,t) \leq v^R(r,s,t)$ when π is a strategy that proposes a running play in state (r,s,t). Thus, in both cases, we have $v^\pi(r,s,t) \leq \max\{v^P(r,s,t), v^R(r,s,t)\}$ and since this inequality holds for every strategy π, in particular it holds when π is the optimal strategy for starting state (r,s,t). Thus $v^*(r,s,t) \leq \max\{v^P(r,s,t), v^R(r,s,t)\}$.

Equations (2)-(4) justify the following dynamic programming algorithm for football strategy.

Initialize dynamic programming tables T^P, T^R, T^* each indexed by elements of
$\{(r,s,t) : 1 \le r \le n, 1 \le s \le k, 1 \le t \le d\} \cup \{\top, \bot\}$
Set $T^*[\top] = T^P[\top] = T^R[\top] = 1$
Set $T^*[\bot] = T^P[\bot] = T^R[\bot] = 0$
for $r = 1, \ldots, n$ **do**
  for $s = 1, 2, \ldots, k$ **do**
    for $t = 1, \ldots, d$ **do**
      $T^P[(r,s,t)] = \sum_{y=0}^{n} p_y \cdot T^*[\textsc{Advance}(y,r,s,t)]$
      $T^R[(r,s,t)] = \sum_{y=0}^{n} r_y \cdot T^*[\textsc{Advance}(y,r,s,t)]$
      $T^*[(r,s,t)] = \max\{T^P[(r,s,t)],\, T^R[(r,s,t)]\}$
    end
  end
end
if $T^P[(n,k,d)] > T^R[(n,k,d)]$ **then**
  return **pass**
else
  return **run**
end

The proof of correctness is by induction on the entries of the dynamic programming tables, in the order in which the algorithm fills in those entries. The induction hypothesis is that T^P, T^R, T^* respectively store the correct values of v^P, v^R, v^* for each element of the index set $\{(r,s,t) : 1 \le r \le n, 1 \le s \le k, 1 \le t \le d\} \cup \{\top, \bot\}$. The base cases $T^*[\top]$ and $T^*[\bot]$ follow directly from the definitions of $v^*(\top)$ and $v^*(\bot)$. In the loop iteration that computes $T^*[(r,s,t)]$, note that the table values $T^*[\textsc{Advance}(y,r,s,t)]$ appearing on the right-hand sides of the equations defining $T^P[(r,s,t)]$ and $T^R[(r,s,t)]$ refer to values already computed in earlier iterations of the nested loops: this is because $\textsc{Advance}(y,r,s,t)$ is always either an element of $\{\top, \bot\}$ or a state (r', s', t') satisfying $r' < r$ or $r' = r$, $s' = s$, $t' < t$; in all cases, $\textsc{Advance}(y,r,s,t)$ refers to the index of a table entry computed strictly earlier than $T^*[(r,s,t)]$. Therefore, the induction hypothesis allows us to equate the values $T^*[\textsc{Advance}(y,r,s,t)]$ occurring on the right side of the formulas defining $T^P[(r,s,t)]$ and $T^R[(r,s,t)]$ with the correspondiong values $v^*(\textsc{Advance}(y,r,s,t))$ appearing on the right side of equations (2) and (3). It follows that $T^P[(r,s,t)] = v^P(r,s,t)$ and $T^R[(r,s,t)] = v^R(r,s,t)$, and finally that

$$T^*[(r,s,t)] = \max\{T^P[(r,s,t)], T^R[(r,s,t)]\} = \max\{v^P(r,s,t),\, v^R(r,s,t)\} = v^*(r,s,t)$$

by equation (4). This completes the induction step.

The running time is $O(n^2kd)$, as can be seen by counting iterations of the nested loops and observing that $O(n)$ operations are performed inside the innermost loop.