(3) (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 $1 \leq i \leq j \leq n$, $A[i, j]$ denotes the probability that A wins in the district $[i, j]$. The probability matrix A 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.

1 Algorithm

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. Thus, our problem is to compute $OPT(n, k)$.
Computing $OPT(r, \ell)$: for some $j \leq r$, let $[j, r]$ be a district in the optimal solution for $OPT(r, \ell)$. Then, using linearity of expectation, it follows that $OPT(r, \ell) = OPT(j-1, \ell-1) + A[j, r]$. Since there are bounds of $s_{\min}$ and $s_{\max}$ on the size of a district, it implies that j must range in $[r - s_{\max} + 1, r - s_{\min} + 1]$ (if this interval is a well-defined interval of precincts). To make the notation more precise, the lower bound should be lower $= \max(r - s_{\max} + 1, 1)$, and the upper bound is upper $= r - s_{\min} + 1$ (we will never go over n).
The recurrence will be the following:

$$\mathrm{OPT}(r, \ell) = \begin{cases} \max\{\mathrm{OPT}(j-1, \ell-1) + A[j, r] : j \in [\text{lower}, \text{upper}] & \text{if } r \geq s_{min} \text{ and } \ell \geq 1 \\ -\infty & \text{if } [\text{lower, upper}] = \emptyset \end{cases}$$

The second case will handle the case when there's no valid partitions. For example, when $s_{\min}$ is larger than $r + 1$, the upper will be negative.

$$\text{The base cases are the following: } \begin{cases} \mathrm{OPT}(0, 0) = 0 \\ \mathrm{OPT}(i, 0) = \mathrm{OPT}(0, j) = -\infty & \text{if } i, j > 0 \\ \mathrm{OPT}(i, 1) = A[1, i] & \text{if } i \in [s_{\min}, s_{\max}] \\ \mathrm{OPT}(i, 1) = -\infty & \text{if } i \in [0, s_{\min} - 1] \cup [s_{\max} + 1, n] \end{cases}$$

The algorithm is the following:

1. Initialize $\mathrm{OPT}(0, 0), \mathrm{OPT}(i, 0), \mathrm{OPT}(0, j), \mathrm{OPT}(i, 1)$ according to the base cases above.
2. for $r = 1$ to n
3. for $\ell = 2$ to k

4. use the recursive relation above to compute $OPT(r, \ell)$.

5. Output $OPT(n, k)$.

Now we do the backtracking. The idea is to loop through all the valid j's in the recurrence $\max\{\text{OPT}(j-1, \ell-1) + A[j, r] : j \in [\text{lower}, \text{upper}]$, and try to find the one we picked before. If the j^* satisfies $\text{OPT}(r, \ell) = \text{OPT}(j^* - 1, \ell - 1) + A[j^*, r]$, then we know j^* is in the optimal solution. In other words, we're trying to find $\arg\max_j \text{OPT}(j-1, \ell-1) + A[j, r]$. The algorithm looks like the following:

1. Initialize D to be an array of size k where $D(i)$ will represent range of precincts for district i.

2. Let $r_{cur} = r$.

3. for $\ell = k$ to 2

4. Let $r_{prev} \in [\text{lower}, \text{upper}]$ be such that $OPT(r_{cur}, \ell) = OPT(r_{prev} - 1, \ell - 1) + A[r_{prev}, r_{cur}]$.

5. Set $D(\ell) = [r_{prev}, r_{cur}]$.

6. Set $r_{cur} = r_{prev} - 1$.

7. Set $D(1) = [1, r_{cur}]$.

2 Proof of Correctness

We'll use induction to prove our algorithm is correct. We induct on l. Our inductive hypothesis $I(\ell)$ is that for all r, $P(r, l)$ is the optimal way to gerrymander first r precincts into l districts.
First we prove the base cases are correct when $l = 0, 1$. We'll do it case by case.

1. When there's no districts and no precincts, the probability that A wins is clearly 0.

2. When we're not allowed to form any districts, or when we don't have any precincts to form any non-zero districts, the probability should be $-\infty$ to represent this case will not happen.

3. When we're partitioning everyone into 1 district, only certain i's are valid since each district requires at least $s_{\min}$ and at most $s_{\max}$ precincts. For those valid i's, the probability of winning is simply $A[1, i]$, meaning they're all in the same district.

4. Similar to the previous case, but now we're setting all invalid partitions to $-\infty$ since these are invalid partitions.

5. Note: Setting invalid/impossible cases to $-\infty$ is necessary in this problem to guarantee the correctness of backtracking since $A[i, j]$ can be 0. If the whole $l-1$ column is 0, we want the algorithm only picks the valid partitions with $A[i, j] = 0$, instead of some other invalid partitions. Since we initialized all invalid base cases $-\infty$, we can make sure our backtracking is correct.

The base cases initialized the first two columns of the DP table.
(Note: the inductive proof mimics K&T pg253)
Then we prove the inductive step: suppose $P(r, l')$ holds for all $l' \in \{1, 2, ..., l-1\}$[1], we want to prove that $P(r, l)$ also holds. Assume we have some optimal solution O for partitioning r precincts into l districts. Since all the precincts must belong to some districts, the r-th precincts must belong to one of its valid partitions, namely all $[j, r]$ such that $j \in [\text{lower}, \text{upper}]$. If we choose the partition $[j, r]$, then we know precincts $1, ..., j$ are partitioned into $l-1$ districts. Given we already have all optimal values for $P(r, l-1)$ for all r, the j that maximizes $\text{OPT}(j-1, \ell-1) + A[j, r]$ must be the optimal partition for $P(r, l)$.
Lastly, we need to prove the correctness of the backtracking. We already proved that our recurrence is correct; therefore, $[j, r]$ belongs to the optimal partition if and only if it satisfies $\text{OPT}(j-1, \ell-1) + A[j, r]$. By our algorithm, we correctly keep track of the current last partitioning and gerrymander the last district as per our recurrence relation.

3 Runtime Analysis

We have nk subproblems in total. In each subproblem, we iterate $c = s_{\max} - s_{\min} + 1$ times; each iteration takes $O(1)$ since getting the value of $\text{OPT}(j-1, l-1)$ takes $O(1)$ and performing addtion also takes $O(1)$. Therefore, the runtime before backtracking is $O(nk * c) = O(nk)$ (either $O(nk)$ or $O(n^k)$ here is fine, it depends on whether you think c is a constant or is bounded by n).
The backtracking algorithm has at most k iterations, and each iteration takes $O(c)$ as we're looping through all valid partitions to find the $\arg\max_j$. Therefore, it takes $O(ck) = O(k)$ or $O(nk)$.
The total runtime is $O(nk + nk) = O(nk)$ or $O(n^2k + nk) = O(n^2k)$.

[1] Actually, this question doesn't require to use strong induction, since we're only looking at the previous column.