1 Digging for Oil: Solution

Solution 1

$O(n^8)$: Try all possible placements of squares, add their interior totals. Such a solution should score around 10%.

Solution 2

$O(n^6)$: Try all possible placements of squares, using precomputed cumulative sums to allow the sum under an area to be determined in O(1). Such a solution should score around 30%.
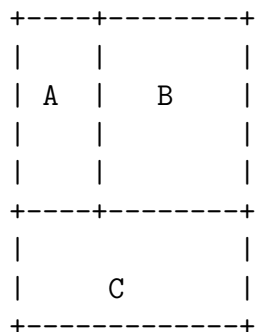
Solution 3

$O(n^2)$: Observe that no matter how you place three squares, it is possible to separate one of them with either a horizontal or vertical cut. For example:
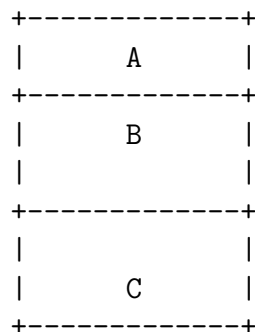
```
XXZZ
XXZZ_____cut
YY           here
YY
```

There are $O(n)$ possible cuts. Without loss of generality, assume the answer is a horizontal cut leaving two pieces in the region above and one in the region below. Of the side with two pieces, these pieces can be separated by either a horizontal or vertical dividing line. This means that the square can be in one of two configurations (or any of the three other rotations of these):

```
+----+--------+         +-------------+
|    |        |         |      A      |
| A  |   B    |         +-------------+
|    |        |         |      B      |
|    |        |         |             |
+----+--------+         +-------------+
|             |         |             |
|     C       |         |      C      |
+-------------+         +-------------+

 Configuration 1         Configuration 2
```

We can precompute the best square in any region that contains a corner square in $O(n^2)$ time. This means that we can determine the best solution for all possible horizontal and vertical cuts in the first configuration as each region shares a corner. To do so, we simple consider all possible values of horizontal and vertical cuts, summing the best solution in each region.

For the second configuration, the same precomputed table can be used to solve for regions A and C. However we cannot use our precomputed table to solve for region B. We additionally need to precompute the best square for each set of k rows, which can be performed in $O(n^2)$. We can then solve for the best solution in the region B by incrementally making region B larger, one row at a time: starting with region B being precisely k rows tall, we can obtain the best solution for region B from our row lookup table. Each time we successively increase the size of region B by one row, we can determine the best solution for region B by taking the maximum of the previous solution for B and the best solution that includes the row just added.

All possible cuts for configurations can be considered in $O(n^2)$ time. Consider all possible cuts for all possible configurations and all four possible rotations of these configurations and output the best solution found.

2 The Siruseri Convention Centre

Consider the intervals as vertices and add an edge between vertices if they overlap. This gives a *interval graph* the problem is to find a lexicographically smallest maximal independent set (MIS) of such an interval graph.

Finding a maximal independent set: Finding a maximum sized candidate is easy, a greedy algorithm solves the suffices. Sort the intervals by increasing order of endpoints and process them in this order. Include an interval if it does not overlap with any of the intervals selected earlier. This can be done in linear time after the sorting.

Lexicographically earliest: $O(N^2)$ Now, To find the the lexicographically least among all maximum independent sets, where the order is defined by the order in which the vertices appear in the input. Greedily add the vertices in the order of their appearance in the input, and find out at each step if the current set can be extended to an independent set of maximum size (this can be done in linear time using the algorithm outlined above.) This gives an O(N^2) algorithm.

Lexicographically earliest: $O(NlogN)$ For the clarity in the explanation, assume that no interval is completely contained in another interval. This assumption can be eliminated with a somewhat more elaborate argument along the same lines and the details are omitted here.

Preliminaries:

- We will think of the intervals as nodes of a graph. Two nodes have an edge if the corresponding intervals intersect. The label of node $[ai, bi]$ is i.

- The required solution is then lexicographically earliest maximum independent set. By MIS(S), where S is a subset of $1, \ldots, n$ we will mean any MIS of the subgraph induced by S.

- Define $right(i) := Min\{j > i \mid i$ and j don't have an edge$\}$.

Observation 1 Our simple solution to (non-lexicographic) MIS shows that

$$\{1, right(1), right(right(1))\ldots\}$$

is a solution.

Subroutine that takes $i < j$ and outputs $|\text{MIS}\{i, i+1, \ldots, j\}|$ in $log(n)$ time: Using the first observation, the size of the MIS of $\{i, i+1, \ldots, j\}$ is $1 + Max\{k : right^k(i) \leq j\}$. This subroutine will essentially guess the bits of the correct k. The data-structure to do this (created in the pre-processing stage) are $log(n)$ arrays $Right[1][-], \ldots, Right[log(n)][-]$ where $Right[k][t] := right^{2^k}[t]$.

Main Program First compute the size of the MIS, $s* = |\text{MIS}\{1, \ldots, n\}|$. Take node $i0$ with earliest time. Neighbors of $i0$ are $\{i, i+1, \ldots, j\}$ for some $i \leq i0 \leq j$. These i, j can be found by binary search in $log(n)$ time. To test whether $i0$ can be included in a MIS, we need to check if $s* = |\text{MIS}\{1, \ldots, i-1\}| + 1 + |\text{MIS}\{j+1, \ldots, n\}|$. This can be done in $log(n)$ time using the subroutine.

Suppose $i0$ passed the test. Take the next node $i1$ (by input ordering)). Now we first locate (in $log(n)$ time through binary search) which of the two sides $\{1, \ldots, i-1\}$ or $\{j+1, \ldots, n\}$ does $i1$ belong to. If it belongs to neither, then $i1$ was a neighbor of $i0$, and so fails the test. Otherwise, suppose $i1$ was in $\{j+1, \ldots, n\}$, and its neighbors were $\{i', i'+1, \ldots, j'\}$ where $j+1 \leq i' \leq i1 \leq j' \leq n$. Now we need to check if $|\text{MIS}\{j+1, \ldots, n\}| = |\text{MIS}\{j+1, \ldots, i'-1\}| + 1 + |\text{MIS}\{j'+1, \ldots, n\}|$.

The data-structure required for this part is a segment tree (or anything that can support insertions, and queries asking smallest element in the data-structure larger than a number given in the query).

3 The Great ATM Robbery: Solution

Decompose the graph into its Strongly connected components (SCCs). Clearly, if Banditji visits any such component, he will rob every node in the component. The set of SCCs forms a DAG and we would like the find the longest path in this resulting DAG (where every node has weight equal to the sum of the reserves at all ATMS in the component) from the component with the city center to the any component containing a pub. This algorithm should run in time linear in the number of edges.