Asia-Pacific Informatics Olympiad 2015

May 9, 2015

Hosted by Indonesia

ANALYSIS

Bali Sculptures

Time limit: 1 s
Memory limit: 64 MB

Subtask 1

Constraints: $1 \leq N \leq 20; 1 \leq A \leq B \leq N; 0 \leq Y_i \leq 1,000,000,000$

Due to the small N of this subtask, it is possible to do a brutal complete search. We can try whether to insert a "separator" between each consecutive numbers. Among all those possible "separator" positions, we only consider those with more than or equal to A and less than or equal to B groups, and find the minimum OR-SUM. The complexity for this solution is $O(N2^N)$.

The solutions for the next subtasks require an understanding of Dynamic Programming.

For simplicity, let $Y = max\{Y_i\}$.

Subtask 2

Constraints: $1 \leq N \leq 50; 1 \leq A \leq B \leq \min(20, N); 0 \leq Y_i \leq 10$

We would like to create a dynamic programming table with three parameter, *curpos*, *curOR*, and *curpartition.*

$dp[curpos][curOR][curpartition]$ will be *true* if and only if there exists a partition of the first *curpos* elements with *curpartition* groups and the OR-sum is *curOR*. It can be computed by this recurrence:

$$dp[curpos][curOR][curpartition] =$$

$$\bigvee_{i=curpos}^{N} dp[i+1][curOR|sum[curpos,i]][curpartition+1]$$

where $sum[x, y]$ is the sum of the ages of sculpture x to y inclusive.

After computing the dp table, we are finding the minimum number X such that there exists P, $A \leq P \leq B$ where $dp[N][X][P]$ is true. We can loop for all possible values for X. *curOR* can be as worse as $O(YN)$. Therefore, the complexity for this solution is $O(NYNNN + NYN) = O(N^4 Y_i)$

Subtask 3

Constraints: $1 \leq N \leq 100; 1 = A \leq B \leq N; 0 \leq Y_i \leq 20$

$A = 1$ means we can try to use as few groups as possible for any fixed *curOR* and *curpos*. We make a minor modification to our solution from subtask 2. We remove the third parameter, left with only *curpos* and *curOR*. For this subtask, $dp[curpos][curOR]$ will return the fewest number of groups to make the partition of the first *curpos* elements with the OR-sum is *curOR*. It can be computed by

$$dp[curpos][curOR] = 1 + \min_{i=curpos}^{N} dp[i+1][curOR|sum[curpos,i]]$$

After computing the dp table, we are finding the minimum number X such that $dp[N][X] \leq B$. The complexity for this solution is $O(NYNN + NY) = O(N^3 Y)$

Subtask 4

Constraints: $1 \leq N \leq 100; 1 \leq A \leq B \leq N; 0 \leq Y_i \leq 1,000,000,000$

We are looking for minimum OR-SUM. We will construct the answer bit-by-bit, from the most significant bit.

It is always optimal if greedily fix the most significant bit of the OR-SUM to be zero, regardless of the other bits. If it is possible, then the most significant bit should be zero; if it is not, it will be one. Then, we consider the next significant bit, and do the same, keeping the current "bit prefix" not changed.

In other words, we iterate k (current bit position) from $\log(YN)$ to 0. For each k, we try whether it is possible to set the k-th bit of the OR-SUM to be zero (ignoring the rest of least significant bits). The check can use the dynamic programming method explained in subtask 2, but we drop the *curOR* parameter. Each group sum now must not change the value of the current bit prefix. The complexity for this solution is $O(\log(YN)NNN) = O(N^3 \log(YN))$

Subtask 5

Constraints: $1 \leq N \leq 2,000; 1 = A \leq B \leq N; 0 \leq Y_i \leq 1,000,000,000$

We can solve this subtask by combining the idea from subtask 4 and subtask 3. From subtask 3, we can drop the *curpartition* parameter by computing the minimum number of partition in the DP table instead. From subtask 4, we can drop the *curOR* parameter. By combining both ideas, we can have a DP table where the parameter is only *curpos*. The complexity for this solution is $O(\log(YN)NN) = O(N^2 \log(YN))$.

Jakarta Skyscrapers

Time limit: 1 s
Memory limit: 256 MB

Subtask 1

Due to the small constraint of this subtask, it is possible to do a breadth-first search, with the current positions of the doges as the state. From a state, there can be at most M^3 transitions, since each doge can either move left, right, or stay. The number of states is M^{N+1}, since each doge can either have the news (N possible positions) or it does not have the news (it must be in its starting building). The complexity of this solution is $O(M^{N+4})$.

Subtask 2

Note that for doge 1 to find the news, there must be a sequence of doges $x_1, x_2, \ldots, x_k$ such that doge x_i got the news from doge x_{i-1}, $x_1 = 0$ as the first doge to get the news, and $x_k = 1$. By using this observation, we can reduce the state to only keep track of one doge.

Now, we can optimize the BFS to only keep track of the position of the current doge, and the power of the current doge. The state becomes (position, power), and the number of vertices is at most NM. From each vertex, we can have two edges:

- Move: Connect $(\mathit{position}, \mathit{power})$ to $(\mathit{position} + \mathit{power}, \mathit{power})$ and $(\mathit{position} - \mathit{power}, \mathit{power})$.
- Move and pass the news to another doge: Connect $(\mathit{position}, \mathit{power})$ to $(\mathit{position}+\mathit{power}, \mathit{new\_power})$ and $(\mathit{position} - \mathit{power}, \mathit{new\_power})$

From each vertex, there can be at most two edges of the first type and M edges of the second type. The total number of edges is $O(N^2M)$. Therefore, the complexity of this solution is $O(NM^2)$.

Subtask 3

We can optimize this further by transforming the problem to a weighted graph. If we consider a series of jump from one doge to another as one single edge, we can reduce the state to just (doge). In other words, connect $\mathit{doge}1$ to $\mathit{doge}2$ with an edge of weight w, if $\mathit{doge}2$ can be reached by $\mathit{doge}1$ in w jumps. In the worst-case scenario, the graph is complete with $O(M^2)$ edges. By running Dijkstra's algorithm on this graph, we get a $O(M^2 \log M)$ solution.

Subtask 4

The problem with the solution of Subtask 2 is that if there are X doges with different powers in building Y, every jump that can reach Y will need to connect with at least X vertices. To reduce this number, we introduce a dummy entry vertex for each building. Every jump that can reach this vertex will connect to this dummy vertex with cost 1, and this dummy vertex will connect to every doges in the building with cost 0. In other words, if we write this new dummy vertex as $(\mathit{position})$,

- Connect $(\mathit{position}, \mathit{power})$ to $(\mathit{position}+\mathit{power}, \mathit{power})$ and $(\mathit{position}-\mathit{power}, \mathit{power})$ with an edge of weight 1.
- Connect $(\mathit{position}, \mathit{power})$ to $(\mathit{position} + \mathit{power})$ and $(\mathit{position} - \mathit{power})$ with an edge of weight 1.
- Connect $(\mathit{position})$ to $(\mathit{position}, \mathit{power})$ for each doge with power power in $\mathit{position}$ with an edge of weight 0.

Now, the number of vertices is still the same (NM), but the number of edges is reduced to N edges from $(\mathit{position})$ vertices, and $2NM$ edges from $(\mathit{position}, \mathit{power})$ vertices. The complexitiy is $O(NM)$.

Subtask 5

We still need to cut the number of vertices for this subtask. To do this, we combine the idea from subtask 3 and subtask 4:

Create $(position, power)$ vertices only for $power \leq \sqrt{N}$. From a $(position)$ vertex, we connect to $(position, power)$ vertex for doges with powers that are less than or equal to $\sqrt{N}$. If the power of the doge is more than $\sqrt{N}$, loop through the possible positions that this doge can jump to, either directly or indirectly, and connect to the entry vertex of those buildings. Now, the number of vertices is at most $O(M\sqrt{N})$.

The number of edges can be analyzed as follows:

- There can only be two edges from a $(position, power)$ vertex connecting it to another (position, power) vertex. The total number of these edges is at most $2N\sqrt{N}$.
- There can be one edge from each $(position, power)$ vertex connecting it to a $(position)$ vertex. The total number of these edges is at most $N\sqrt{N}$.
- The number of edges from a $(position)$ vertex to a $(position, power)$ vertex is at most $\sqrt{N}$ for each position, since there are only $\sqrt{N}$ different values of power it can connect to. The total number of these edges is at most $N\sqrt{N}$.
- The number of edges from a $(position)$ vertex to another $(position)$ vertex is bounded by the number of jumps for each doge. Each doge with power greater than $\sqrt{N}$ can only jump at most $\sqrt{N}$ times. The total number of these edges is at most $M\sqrt{N}$.

Therefore, the number of edges is $O((M + N)\sqrt{N})$.

Running Dijkstra's algorithm in this graph gives a $O((M + N)\sqrt{N}\log(M + N))$ solution.

Palembang Bridges

Time limit: 2 s
Memory limit: 256 MB

Subtask 1

Ignore the paths which don't cross the river (i.e., compute them separately).

Also, ignore the length of the bridge (i.e., compute it separately).

It can be proved that the solution is optimal when the bridge is built at one of the positions S_is or T_is. Therefore, we can choose the position by brute-force. The complexity will be $O(N^2)$.

Subtask 2

If we build a bridge at position X, then D_i (driving distance of citizen i) will be $|X - S_1| + |X - T_1|$. The total sum is $S = |X - S_1| + |X - T_1| + \cdots + |X - S_N| + |X - T_N|$.

Sort $\{S_1, T_1, \ldots, S_N, T_N\}$ in non-decreasing order, and let the sorted sequence be $\{x_1, x_2, \ldots, x_{2N}\}$. Then, $S = |X - x_1| + |X - x_2| + \cdots + |X - x_{2N}|$.

It can be proved that S is minimized when $x_N \leq X \leq X_{N+1}$. Then, S will be $x_{2N} + x_{2N-1} + \cdots + x_{N+1} - x_N - \cdots - x_2 - x_1$.

The complexity of this solution is $O(N \log N)$.

Subtask 3

Similar to Subtask 1, it can be proved that the solution is optimal when the bridge is built at two of the positions S_is or T_is. Therefore, we can choose the positions by brute-force. The complexity will be $O(N^3)$.

Subtask 4

Suppose we build two bridges at positions L and R (wlog $L < R$).

For a particular citizen i, let $P_s = min(S_i, T_i) and P_e = min(S_i, T_i)$. Note that we only consider citizen who has to cross the river to go from home to the office. (P_s, P_e) now can be considered an interval.

For a given interval (P_s, P_e) (again wlog $P_s < P_e$), how do we determine which bridge to use? We can only bridge L iff

$$|P_s - L| + |P_e - L| < |P_s - R| + |P_e - R| \tag{1}$$

(If they are equal, we can use either bridge).

Using Equation 1, we will now prove the following

Lemma 0.0.1. *It is optimal to use the bridge at L if*

$$P_s + P_e < L + R \tag{2}$$

And it is optimal to use the bridge at R if

$$P_s + P_e > L + R \tag{3}$$

Note that this does not imply the invese. That is, optimality of using bridge at L or R does not imply the equations.

Proof. To prove Lemma 0.0.1, we need to prove that we can derive Equation 1 from Equation 3. We do so case by case. Denote the interval (P_s, P_e) with

```
|---------|
```

Case 1

```
|---------| L R
```

That is, we have $P_e \leq L$.

In this case, Equation 1 expands to

$$\begin{aligned} |P_s - L| + |P_e - L| &< |P_s - R| + |P_e - R| \\ 2L - (P_s + P_e) &< 2R - (P_s - P_e) \\ 2L &< 2R \end{aligned}$$

Which is always true. So, we need to prove that in this case, $P_s + P_e < L + R$ always holds. Since we have $P_e \leq L$:

$$\begin{aligned} P_s + P_e &\leq P_e + P_e && (\text{Since } P_s \leq P_e) \\ &\leq 2L && (\text{Since } P_e \leq L) \\ &< L + R && (\text{Since } L < R) \end{aligned}$$

Case 2

```
L R |---------|
```

That is, we have $P_s \geq R$. We prove similar to Case 1. Equation 1 expands to

$$\begin{aligned} |P_s - L| + |P_e - L| &< |P_s - R| + |P_e - R| \\ (P_s + P_e) - 2L &< (P_s - P_e) - 2R \\ 2L &> 2R \end{aligned}$$

Which is always false. So, we need to prove that in this case, $P_s + P_e < L + R$ never hold. Since we have $P_s \geq R$:

$$\begin{aligned} P_s + P_e &\geq P_s + P_s && (\text{Since } P_s \leq P_e) \\ &\geq 2R && (\text{Since } P_s \geq R) \\ &> L + R && (\text{Since } L < R) \end{aligned}$$

Case 3

```
|---------|
L         R
```

Thus, we have $L \leq P_s \leq R \leq P_e$.

Equation 1 expands to

$$\begin{aligned}
|P_s - L| + |P_e - L| &< |P_s - R| + |P_e - R| \\
(P_s + P_e) - 2L &< R - P_s + P_e - R \\
(P_s + P_e) - 2L &< P_e - P_s \\
2P_s &< 2L
\end{aligned}$$

Which is always false. So, we need to prove that in this case, $P_s + P_e < L + R$ never hold:

$$\begin{aligned}
P_s + P_e &\geq L + P_e \quad (\text{Since } L \leq P_s) \\
&\geq L + R \quad (\text{Since } R \leq P_e)
\end{aligned}$$

Case 4

```
|---------|
L         R
```

Thus, we have $P_s \leq L \leq P_e \leq R$.

Equation 1 expands to

$$\begin{aligned}
|P_s - L| + |P_e - L| &< |P_s - R| + |P_e - R| \\
L - P_s + P_e - L &< 2R - P_s - P_e \\
2P_e &< 2R
\end{aligned}$$

Which is always true. So, we need to prove that in this case, $P_s + P_e < L + R$ always hold:

$$\begin{aligned}
P_s + P_e &\leq L + P_e \quad (\text{Since } P_s \leq L) \\
&\leq L + R \quad (\text{Since } P_e \leq R)
\end{aligned}$$

Case 5

```
|---------|
L   R
```

Thus, we have $P_s \leq L \leq R \leq P_e$.

Equation 1 expands to

$$\begin{aligned}
|P_s - L| + |P_e - L| &< |P_s - R| + |P_e - R| \\
L - P_s + P_e - L &< R - P_s + P_e - R \\
0 &< 0
\end{aligned}$$

Which is always false, and in particular, it implies that the distance of using either of the bridges is the same. Thus, using any of the two bridges is always optimal.

Case 6

```
|---------|
L                   R
```

Thus, we have $L \le P_s \le P_e \le R$.

From Equation 3:

$$\begin{aligned} P_s + P_e &< L + R \\ 2(P_s + P_e) &< 2L + 2R \\ (P_s + P_e) - 2L &< 2R - (P_s + P_e) \\ |P_s - L| + |P_e - L| &< |P_s - R| + |P_e - R| \end{aligned}$$

Which is equation 1.

□

Thus, there exists a way to separate the intervals into those which should go to the left bridge and those which should go to the right bridge. Sort the intervals by $P_s + P_e$, and for every possible separation of this sequence of intervals, try to find the optimal placement of two bridges.

The optimal placement of a partition can be solved using the algorithm in Subtask 2. So, the total complexity is $O(N^2 \log N)$.

Subtask 5

One way to do this in $O(N \log^2 N)$: we keep track of a data structure to count the cost of placing a bridge in each possible position (initially, all are zero). We add the intervals one by one. When we add an interval (P_s, P_e), we are actually adding to each position x, $|P_s - x|$ and $|P_e - x|$. $f(x) = |A - x|$, though, is convex, so the sum over all these convex functions is a convex function. So we can ternary search over the sum of these convex functions to find the minimum in $\log^2(N)$ time.

There is another solution in $O(N \log N)$. Note that the optimal location of a bridge in each prefix is between the medians of the positions. So, the problem reduces to finding medians of each prefix. This *incremental medians* problem can be solved using using two heaps: one min-heap, and one max-heap. When we add an interval, we add P_s and P_e in the heaps in such a way that both heaps contain equal number of elements (possibly transferring elements between heaps). Then, the medians can be found in $O(1)$.