Problem 6

> Refs:
>
> None

(1)

- No assumption: 98141210
- Under assumption 3: 981120

---

(2)

Algorithm

1. Sort P_i from large to small.
2. Concatenate P_i in this sorted order, first element in the leftmost place.

Proof

Suppose S is an optimal solution, S_x is the x-th digit (from left to right).

Assume that $\exists\ i, j$ such that $i < j$ and $S_i < S_j$, because P_i is exactly one digit long, we can swap S_i, S_j and get S'. Since $S'_i = S_j > S_i$ and $S'_x = S_x\ \forall\ 1 \le x < i, S' > S$. This contradicts with the assumption that S is an optimal solution, therefore $\forall\ i < j,\ S_i \ge S_j$.

Time Complexity

Time complexity of each step:

1. Sort in $O(n \log n)$.
2. Concatenate in $O(n)$.

Total time complexity $O(n \log n) + O(n) = O(n \log n)$.

---

(3)

Algorithm

1. Sort P_i by:
    1. Compare their first (leftmost) digit, the larger one goes first.
    2. If the same, compare their next digit. If there are no second digit, compare as if it's the same as first digit. The larger one does first.
    3. If they are still not separated, repeat previous step until all digits are compared.
    4. If all digits are compared, their order will remain the same.

2. Concatenate P_i in this sorted order, first element in the leftmost place.

Proof

Suppose S is an arbitrary arrangement, S_x is the x-th preference value (from left to right) concatenated, $S_x[d]$ is the d-th digit from left to right.

Assume that $\exists\, i, j$ such that $i < j$ but S_i should be after S_j according to the compare method above.

First consider when $j = i + 1$, that is, when they are next to each other. S_i and S_j should have a common prefix of length $l \geq 0$. There are three cases:

- If $l < \min\{len(S_i), len(S_j)\}$, then $S_i[l+1] < S_j[l+1]$. Let the number after swapping be S'. Because S and S' are the same until $S_i[l]$, since $S'_i[l+1] = S_j[l+1] > S_i[l+1]$, $S' > S$.
- If $l = len(S_i)$, then $l > 0$ and $S_j[l+1] > S_i[1] = S_j[1]$. Let the number after swapping be S'. Because S and S' are the same until $S_i[l]$, since $S'_i[l+1] = S_j[l+1] > S_i[1] = S_j[1]$, $S' > S$.
- If $l = len(S_j)$, then $l > 0$ and $S_i[l+1] \leq S_i[1] = S_j[1]$. Let the number after swapping be S'. Because S and S' are the same until $S_i[l]$, since $S'_j[1] = S_i[1] \geq S_i[l+1]$, $S' \geq S$.

In all three cases, any unordered neighboring pair gives a higher S after swapping. Only if they are sorted by the compare method above, there will be no unordered neighboring pair. Therefore this sorting yields the maximum satisfying value.

Time Complexity

Time complexity of each step:

1. The compare function runs in $O(1)$ time (because P_i is capped at 1000), therefore the sort runs in $O(n \log n) \cdot O(1) = O(n \log n)$ time.
2. Concatenating runs in $O(n)$ time.

Total time complexity $O(n \log n) + O(n) = O(n \log n)$.

---

(4)

Algorithm

1. Let $m = 0$.
2. Loop through P_i to record how many times each digit has appeared as a_{digit}. Also let $b = a_1 + a_4 + a_7, c = a_2 + a_5 + a_8$.
3. Let $b' = b \bmod 3, c' = c \bmod 3$.
   - If $b' > c'$, remove the least $b' - c'$ digits that satisfied $d \equiv 1 \pmod 3$ and change the corresponding a_d.
   - If $c' > b'$, remove the least $c' - b'$ digits that satisfied $d \equiv 2 \pmod 3$ and change the corresponding a_d.
4. Loop d from 9 to 0, concatenate a_d digits of d to back of m.
5. m is the maximum satisfying number under assumption 1 & 3.

Proof

According to subproblem (2), larger digits should be on the left. To make it divisible by 3, if d is not divisible by 3, group them by their remainder. From each group, repeatedly choose the largest 3 out. If some digits are left, choose one $d \equiv 2 \pmod 3$ with one $d \equiv 1 \pmod 3$.

Because the number of digits is maximized and larger digits are on the left, m is maximized.

Time Complexity

Time complexity of each step:

1. $O(1)$
2. $O(n)$
3. $O(1)$
4. $O(\sum a_d) = O(n)$
   - All cases are $O(a_d)$.

Total time complexity is $O(1) + O(n) + O(1) + O(n) = O(n)$.

---

(5)

98653

---

(6)

Algorithm

Let $f(A, k)$ be the maximum satisfying value of preference values A_i with k courses selected.
$merge(x, y)$ is the maximum value of merging x and y in to a single number, while preserving order of digits from the same source.

The maximum satisfying value given P_i and M_i is $\max_{0 \le j \le k}(merge(f(P, j), f(M, k-j)))$.

$f(A, k)$ is done by:

1. Let $r = 0, s = 0$.
2. Loop i from 1 to k.
   1. Choose the largest digit from $A[1 : 1 + len(A) - k - s]$, concatenate the digit d to the back of r.
   2. Increase s by the number of elements in front of d. Remove d and every digit in front of it from A.
3. $r = f(A, k)$

$merge(x, y)$ is done by:

1. Let $r = 0$.
2. While x and y are not empty, choose the larger first bit and concatenate it to the back of r, then remove that digit from the source number.

3. Concatenate all digits left to the back of r.
4. $r = merge(x, y)$.

Proof

$f(A, k)$

This algorithm tries to maximize the first digit, then the second digit and so on.

Suppose an optimal solution S has $S[1] \neq \max A[1 : 1 + len(A) - k]$, because we can remove only up to $len(A) - k$ elements, $S[1] \in A[1 : 1 + len(A) - k]$. Therefore $S[1] < \max A[1 : 1 + len(A) - k]$, and choosing $\max A[1 : 1 + len(A) - k]$ for $S[1]$ should be the optimal solution. Other digits can be proved using the same method.

$merge(x, y)$

If during the merge process, when comparing the first digit left, the smaller one is chosen first, it must not be the optimal solution. Because choosing the larger one first gives a larger answer.

$\max_{0 \leq j \leq k}(merge(f(P, j), f(M, k - j)))$

The maximum satisfying value must be consist of j values from P and $k - j$ values from M, therefore we chould choose the maximum from each case of j.

Time Complexity

$f(A, k)$

In each step:

1. $O(1)$
2. $O(k) \cdot O(n)$
    1. $O(n)$
    2. $O(n)$

Total time complexity is $O(kn)$.

$merge(x, y)$

Total time complexity is $O(len(x) + len(y))$.

$\max_{0 \leq j \leq k}(merge(f(P, j), f(M, k - j)))$

Total time complexity is:

$$\begin{aligned}
&\sum_{0 \leq j \leq k} O(jn) + O((k - j)n) + O(j + (k - j)) \\
=&\sum_{0 \leq j \leq k} O(kn) + O(k) \\
=&\sum_{0 \leq j \leq k} O(kn) \\
=&O(k^2 n) \\
=&O(kn^2) \text{ (because } k < n)
\end{aligned}$$