Code Jam 2022 - Round 1C

Analysis: Letter Blocks

Let us first notice that this problem is equivalent to finding an order in which partial strings should be concatanted such that occurrences of the same letter appear together.

Test Set 1

Since $\mathbf{N}$ is at most 6, the number of permutations of these strings is at most $6! = 720$.

We can therefore generate all permutations and for each of them we need to verify the final string which results of concatenating the input strings in that particular order.

Let us take a look at the string `CCCABDAEEF`. To verify the string it is enough to:

- Get a set of letters: {A, B, C, D, E, F}
- Create a grouped represantation of a string: `"CABDAEF"`
- The string is good if and only if the lengths coincide.

If for at least one permutation the size matches, we can print out this string. Otherwise, it's impossible.

The time complexity of this solution is $O(\mathbf{N!} \times \sum_{i=1}^{N} |\mathbf{S_i}|)$, which means $\mathbf{N!}$ times the sum of the lengths of the input strings, because there are $\mathbf{N!}$ permutations and the verification of the final string takes linear time.

Test Set 2

In this test set $\mathbf{N}$ can be 100, so $\mathbf{N!}$ is too large to enumerate all permutations.

First of all, we can verify that each of the strings $\mathbf{S_i}$ meets the requirements from the task. This can be done by applying the verification method described in the section above for each $\mathbf{S_i}$ individually. If the verification fails for one of the input strings, then it will certainly fail for any permutation of them and therefore we output `IMPOSSIBLE`.

Let us call middle letters all letters other than the first and last consecutive segment of letters. Next, let us notice that if the string $\mathbf{S_i}$ has more than 2 distinct letters, then:

- If a letter is a middle letter in more than one input string, then those occurrences will not be together in the final string regardless of the order in which we concatenate them. Therefore, this case is impossible.
- If the middle letters exist in a single input string, they don't influence the outcome as those occurrences will be together in the final string regardless of the order in which we concatenate. Therefore, in this case we can assume the input string is simply two letters long removing everything except the first and last letter of it.
Therefore, for each middle letter we can count in how many strings it appears and if the answer is more than 1 for any of them, we can print out `IMPOSSIBLE`.

Since we also verified each string already, we know that each letter appears only in 1 block inside each $\mathbf{S_i}$.

After this step the problem is now simplified into strings of two forms:

- X: Represents the string consisting of only one block of letter X.
- XY: Reperesents the string starting with a block of X letters and ending with a block of Y letters.

If there are two strings of the form X for the same letter, we can concatenate them as they can't be separated by other strings in the final solution. If there are two strings of the form X_1Y_1 and X_2Y_2, then the answer is `IMPOSSIBLE` if $X_1 = X_2$ (due to Y_1) or $Y_1 = Y_2$ (due to X_2), because it means that in any ordering, there would be at least one block of a different letter between letters X_1 and Y_1.

Therefore for each string $\mathbf{S_i}$, we can create the following mappings using sets:

- If $\mathbf{S_i}$ is of the form X, then insert $\mathbf{S_i}$ into $single[X]$.
- If $\mathbf{S_i}$ is of the form XY, then insert $\mathbf{S_i}$ into both $starts[X]$ and $ends[X]$.

In other words, $single[X]$, $starts[X]$ and $ends[X]$ must all contain exactly 1 element for each letter X. If the element already exists, we return `IMPOSSIBLE`.

Starting string

Let us consider starting string as the input string which is not forced by any previous strings in the final answer. When can the given string $\mathbf{S_i}$ be a starting string?

- If $\mathbf{S_i}$ is of the form X, then there must be no other strings ending with letter X, i.e. $ends[X] = \mathbf{S_i}$.
- If $\mathbf{S_i}$ is of the form XY, then $starts[X] = \mathbf{S_i}$ and $ends[X] = null$ and $single[X] = null$.

With these two conditions, we consider a set of candidates C containing all such starting strings.

Extending the block

Let us consider we already built the partial answer A which ends with letter c. If there exists a string at $single[c]$, it is the last chance to append it, because otherwise it would be separated by at least one block of another letter. Similarly, if there exists a string at $starts[c]$, we must extend it now for the same reason.

How to choose the starting string?

It turns out that for starting a new block, we can choose an arbitrary candidate from the candidates set.

$Proof$: Let us assume that we picked string $\mathbf{S_i}$ as the starting string but the optimal solution started the block with $\mathbf{S_j}$. Let us consider the swapped optimal solution in which we swap these blocks.

Let $\mathbf{S_i}$ start with letter a and $\mathbf{S_j}$ with letter b. Then:

- Optimal = |b..X|a....Y|
- Optimal (swapped) = |a....Y|b..X|

Let us consider what happens after swapping:

- Middle letters of these blocks: Any letters between b and a can't be after a and any letters after a can't be before a. Therefore after swap they also remain fine.
- Letters b: Optimal solution can't have any b letters after a. Therefore this swap is okay.
- Letters a: Since $\mathbf{S_i}$ belongs to the candidate set, therefore $ends[a] = \mathbf{S_i}$ or $ends[a] = null$. It means, that there are either no strings ending in a or $\mathbf{S_i}$ is the only string ending with a. Therefore X can't end with a and the swap remains correct.

Final solution

Repeat:

- 1. Pick an arbitrary string from the candidate's set and start the block with it.
- 2. Let e = last letter of the current solution. If starts[e] != null, add starts[e] to current solution and repeat step 2. Otherwise goto step 1.
- 3. If candidate's set is empty, print solution. Otherwise, print `IMPOSSIBLE`.

Time complexity: $O(\mathbf{N} \times \sum_{i=1}^{N} |\mathbf{S_i}|)$, since we are touching each candidate only once.

Code Jam 2022 - Round 1C

Analysis: Squary

The multinomial expansion for the power of 2 is the key to solving this problem. The expansion of the square of the sum of elements $X_1, X_2, \ldots, X_n$ in the list X looks like:

$$\begin{aligned}\text{square of sum} &= (X_1 + X_2 + X_3 + \ldots + X_N)^2 \\ &= X_1^2 + X_2^2 + X_3^2 + \ldots + X_N^2 + 2 \cdot X_1 \cdot X_2 + 2 \cdot X_2 \cdot X_3 + 2 \cdot X_1 \cdot X_3 + \ldots + 2 \cdot X_{N-1} \cdot X_N \\ &= \text{sum of squares} + 2 \cdot \text{sum of pairwise products}\end{aligned}$$

Let $S(X)$ be the sum of elements, $SQ(X)$ be the sum of squares of elements, and $SP(X)$ be the sum of all pairwise products of elements of the list X. We can now rewrite the above equation as:

$$S(X)^2 = SQ(X) + 2 \cdot SP(X)$$

We can also observe the following about how the above values change when an additional element n is added to the list X:

$$\begin{aligned}S(X + [n]) &= S(X) + n \\ SQ(X + [n]) &= SQ(X) + n^2 \\ SP(X + [n]) &= SP(X) + n \cdot S(X)\end{aligned}$$

Our task is to achieve $S(E')^2 = SQ(E')$, where E' is the extended list that we get by adding extra elements to $\mathbf{E}$. In other words, we want to make $SP(E') = 0$.

Test Set 1: $\mathbf{K} = 1$

If we are allowed only a single addition, we must choose an element n such that $SP(\mathbf{E} + [n]) = 0$.

$$\begin{aligned}& SP(\mathbf{E} + [n]) = 0 \\ \implies & SP(\mathbf{E}) + n \cdot S(\mathbf{E}) = 0 \\ \implies & n \cdot S(\mathbf{E}) = -SP(\mathbf{E})\end{aligned}$$

If $S(\mathbf{E}) \neq 0$, we can get a squary list whenever $-SP(\mathbf{E})/S(\mathbf{E})$ is an integer, which happens if and only if $S(\mathbf{E})$ divides $SP(\mathbf{E})$. In that case, $-SP(\mathbf{E})/S(\mathbf{E})$ is our answer.

If $S(\mathbf{E}) = 0$, then $S(\mathbf{E} + [n]) = n$. Since we want $S(\mathbf{E} + [n])^2 = SQ(\mathbf{E} + [n])$, we need $SQ(\mathbf{E} + [n]) = n^2$. This is possible only if $SQ(\mathbf{E}) = 0$, that is, if all elements in $\mathbf{E}$ are zeros. In this case, we can choose any value as our answer. But if any element in $\mathbf{E}$ is not zero, it is impossible to get a squary list with only one addition.

Test Set 2: $\mathbf{K} > 1$

At first, the search space might seem hopelessly broad here. But we can observe (or surmise and then confirm) that it is always possible to get a squary list by adding only two elements:

- $n_1 = 1 - S(\mathbf{E})$
- $n_2 = -SP(\mathbf{E} + [n_1])$

After adding n_1, we have

$$S(\mathbf{E} + [n_1]) = 1$$

After adding n_2, we have

$$\begin{aligned}SP(\mathbf{E} + [n_1, n_2]) &= SP(\mathbf{E} + [n_1]) + n_2 \cdot S(\mathbf{E} + [n_1]) \\ &= SP(\mathbf{E} + [n_1]) + (-SP(\mathbf{E} + [n_1])) \cdot 1 \\ &= 0\end{aligned}$$

Thus, the two numbers satisfy the condition $SP(E') = 0$. We can also see that since the numbers in the original list are each of magnitude no greater than 10^3, $|n_1| \leq 10^6 + 1$, and $|n_2| \leq 2 \cdot 10^{12}$, both well within the limits.

Code Jam 2022 - Round 1C

Analysis: Intranets

We use terminology for graphs. The vertices are the $\mathbf{M}$ machines and the edges are the $\binom{\mathbf{M}}{2}$ links.

Test Set 1

Let's consider the process of assigning priorities to the edges one by one, from the highest priority to the lowest. For simpicity, we number the i-th highest priority as priority i so that the smaller the number, the higher the priority.

Suppose we have assigned the highest i priorities to i edges, and we need to assign the priority $(i+1)$ to a new edge. How can we do?

We have three types of choices. Suppose the new edge with the priority $(i+1)$ is (u, v). Let S_i be the set of vertices that the edges with highest i priorities connect with.

1. If the vertices u and v don't occur in the previously assigned edges. I.e. $u \notin S_i, v \notin S_i$, the number of such edges is $\binom{\mathbf{M}-|S_i|}{2}$. In this case, u and v form a new intranet. So, the number of intranets increase by 1, and $|S_{i+1}|$ equals $|S_i| + 2$.
2. If $u \in S_i, v \notin S_i$, the number of such edges is $|S_i| \cdot (\mathbf{M} - |S_i|)$. In this case, v joins the intranet in which u is located. So, the number of intranets does not change, and $|S_{i+1}|$ equals $|S_i| + 1$.
3. If $u \in S_i, v \in S_i$, the number of such edges is $\binom{|S_i|}{2} - i$. In this case, the edge is not activated. So, the number of intranets do not change, and $|S_{i+1}|$ equals $|S_i|$.

Then, we can use a dynamic programming approach to solve Test Set 1. Let $\mathrm{dp}(i, j, k)$ denote the probability that we assign the highest i priorities to the edges, the set of vertices introduced by these i edges has the size of j, and k intranets have been formed. The transition function can be deduced from the discussion above. The time complexity is $O(\mathbf{M}^4)$ and it is enough to pass Test Set 1.

We can speed this up to $O(\mathbf{M}^2)$ by dropping i from the keys. Let $\mathrm{dp}(j, k)$ denote the probability that after assigning the highest i priorities for some i, the set of vertices introduced by these i edges has the size of j, and k intranets have been formed. The transition is to assign the highest priority **among the edges of the types 1 and 2 above**. The probability of there being a new intranet is $\frac{\binom{\mathbf{M}-|S_i|}{2}}{\binom{\mathbf{M}-|S_i|}{2} + |S_i| \cdot (\mathbf{M} - |S_i|)}$ as we are no longer interested in priorities of the edges of the type 3 above.

Observing the graph

From the solution for Test Set 1, we see that each intranet corresponds to a pair of vertices (u, v) such that both u and v activate the edge (u, v). This fact can also be proved directly, and we describe here.

Let's fix an assignment of priorities and consider the directed graph where the vertices are the machines and the edges are (u, v) such that machine u uses the links connecting machines u and v. Since this graph is a functional graph (each vertex has outdegree 1), each connected component contains exactly one cycle (and possibly some chains of nodes leading into the cycle).

A crucial observation is that the length of a cycle cannot be 3 or more. Assume the contrary: that vertices $u_1, \ldots, u_c$ ($c \geq 3$) form a cycle in this order, and let the priority of the link connecting u_i and u_{i+1} be p_i (indices are modulo c). Note that those links are distinct. Since machine u_i uses the link with highest priority, we have $p_i < p_{i-1}$. This gives us $p_1 > p_2 > \cdots > p_c > p_1$, a contradiction. As a self loop is also impossible, we conclude that every cycle in the graph has length 2.

Test Set 2

Let's call the set of edges activated by both of their endpoints an *active matching* as they form a matching. Now the problem is to compute the probability that the size of the active matching is exactly $\mathbf{K}$.

For a matching X, let $f(X)$ be the probability that the active matching is X. Since it is not easy to compute $f(X)$ directly, we apply inclusion-exclusion principle technique and so let $g(X)$ be the probability that the active matching **contains** X, that is, $g(X) = \sum_{Y \supseteq X} f(Y)$. By inverting this, we have $f(X) = \sum_{Y \supseteq X} (-1)^{|Y|-|X|} g(Y)$. Our answer is then calculated as follows:

$$\sum_{|X|=\mathbf{K}} f(X) = \sum_{|X|=\mathbf{K}} \sum_{Y \supseteq X} (-1)^{|Y|-|X|} g(Y) = \sum_{|Y| \geq \mathbf{K}} \binom{|Y|}{\mathbf{K}} (-1)^{|Y|-\mathbf{K}} g(Y) = \sum_{i \geq \mathbf{K}} \binom{i}{\mathbf{K}} (-1)^{i-\mathbf{K}} \sum_{|X|=i} g(X)$$

All that remains is to compute $g(X)$. Of course, this depends only on $|X|$, and we want to compute it when $|X| = i$, for each $i = \mathbf{K}, \mathbf{K}+1, \ldots, \mathbf{M}$. There are $i!$ possible orders of priorities assigned to X. Let's fix one of them and let the edges in X be $(u_1, v_1), (u_2, v_2), \ldots, (u_i, v_i)$ from the lowest priority to the highest. Then the condition that the active matching contains X is equivalent to, for each $j = 1, 2, \ldots, i$, edge (u_j, v_j) has the highest priority among edges touching at least one of $u_1, v_1, \ldots, u_j, v_j$. Hence we have

$$g(X) = i! \prod_{j=1}^{i} \frac{1}{\binom{\mathbf{M}}{2} - \binom{\mathbf{M}-2j}{2}}.$$

Here we note that the denominators can be factorized to see that they are not divisible by $10^9 + 7$. The number matchings of size i is $\frac{1}{i!2^i} \frac{\mathbf{M}!}{(\mathbf{M}-2i)!}$, and this completes the $O(\mathbf{M})$ time solution (the divisions can be done efficient by using the factorization, though this is not necessary).

In fact, we can find the answers for $K = 1, \ldots, \lfloor \frac{\mathbf{M}}{2} \rfloor$ at the same time in $O(\mathbf{M} \log \mathbf{M})$ time: The transformation from g to f can be represented by a convolution and we can use FFT.