6A Contest

July 12, 2019

Ilya Zban

Discover Vladivostok 2019

A. Bermutation

In this problem we have a permutation p, the number b and we are allowed to make the next operation: take i such that $i \geqslant 1; i + 2b - 1 \leqslant n$, and swap p_i with p_{i+b}, p_{i+1} with p_{i+1+b}, $\ldots$, p_{i+b-1} with p_{i+2b-1}.

We are asked to find the number of permutation p in the lexicographically ordered list of all permutations which can be obtained by the described operations, modulo 120 586 241.

A. Bermutation

First, let's find a number of reachable permutations.

A. Bermutation

First, let's find a number of reachable permutations.

We can see that after applying given operation, positions of elements in permutation modulo b don't change, so the position of element at i-th position in initial permutation can become $i + kb$ for some integer k.

A. Bermutation

First, let's find a number of reachable permutations.

We can see that after applying given operation, positions of elements in permutation modulo b don't change, so the position of element at i-th position in initial permutation can become $i + kb$ for some integer k.

Another observation is that our operation changes the number of inversions in each slice of our permutation (elements with indices $i, i + b, i + 2b, \ldots$) modulo b exactly by 1, so we can determine the oddity of inversions numbers in each slice if we know oddity of inversions number in one slice.

A. Bermutation

First, let's find a number of reachable permutations.

We can see that after applying given operation, positions of elements in permutation modulo b don't change, so the position of element at i-th position in initial permutation can become $i + kb$ for some integer k.

Another observation is that our operation changes the number of inversions in each slice of our permutation (elements with indices $i, i + b, i + 2b, \ldots$) modulo b exactly by 1, so we can determine the oddity of inversions numbers in each slice if we know oddity of inversions number in one slice.

These two observations give us an upper bound to the number of achievable permutations: we can achieve no more than $2^{-(b-1)} \prod_{i=0}^{b-1} (\lceil \frac{n+i}{b} \rceil)!$ different permutations.

A. Bermutation

We can prove that this bound is achievable: if $n \leqslant 2b$, we can put any element from positions $0, b, \ldots,$ on a first place and proceed to $n-1$. When we have $n = 2b-1$, there are exactly $b-1$ remainders modulo b with only one from two potentially achievable permutations. So, we can get exactly $\prod_{i=0}^{b-1} (\lceil \frac{n+i}{b} \rceil)!$ (almost permutation at each slice modulo b) divided by 2^{b-1} (the number of permutations that we can't get).

A. Bermutation

Now using the obtained formula we can calculate the number of given permutation. We can fix the position j of the first different element between our permutation and some lesser achievable permutation, count the number l of smaller elements that we can put at the position j, and add the $l \cdot 2^{-(b-1)} \prod_{i=0}^{b-1} (\lceil \frac{n-j-1+i}{b} \rceil)!$ to the answer.

A. Bermutation

Now using the obtained formula we can calculate the number of given permutation. We can fix the position j of the first different element between our permutation and some lesser achievable permutation, count the number l of smaller elements that we can put at the position j, and add the $l \cdot 2^{-(b-1)} \prod_{i=0}^{b-1} (\lceil \frac{n-j-1+i}{b} \rceil)!$ to the answer.

l can be calculated by keeping Fenwick tree in each slice, and the product $\cdot 2^{-(b-1)} \prod_{i=0}^{b-1} (\lceil \frac{n-j-1+i}{b} \rceil)!$ can be recalculating when we move j, because only one multiplier changes value.

The task is solved in $\mathcal{O}(n \log n)$ time.

B. Grid

In this problem we have a grid with a person looking in some direction in each cell of grid. We need to rotate persons minimal number of times so that there would be no two persons looking at each other.

B. Grid

Let's make a bipartite graph:

B. Grid

Let's make a bipartite graph:

- with nm vertices in left part, corresponding to people in cells;

B. Grid

Let's make a bipartite graph:

- with nm vertices in left part, corresponding to people in cells;
- with $4nm - (n-1)(m-1)$ vertices in right part, corresponding to all centers of each edge in grid;

B. Grid

Let's make a bipartite graph:

- with nm vertices in left part, corresponding to people in cells;
- with $4nm - (n-1)(m-1)$ vertices in right part, corresponding to all centers of each edge in grid;
- edges between vertices from left and right parts if the corresponding cell and edge are adjacent. The cost of edge should be equal to one of $\{0, 1, 2\}$ depending on the cost of turning a person in cell to look in this direction.

B. Grid

The maximal matching of minimal weight in this graph is exactly the optimal arrangement in the grid. There would be no two persons looking at each other, because each person will be directed somewhere, there would be no two persons looking at each other (no more than one man look at each side), and the cost of such arrangement is minimal.

Solution runs in $\mathcal{O}((nm)^2 \log nm)$ if we solve this problem by min cost max flow algorithm with Dijkstra.

C. Heap

In this problem we need to find a number of given d-ary heap in a set of all possible d-ary heaps.

C. Heap

Let's look at the $\mathcal{O}(n^3)$ solution. We want to find the number of smaller d-ary heaps, so we can fix the position p and the value v of first different element in $\mathcal{O}(n^2)$.

C. Heap

Let's look at the $\mathcal{O}(n^3)$ solution. We want to find the number of smaller d-ary heaps, so we can fix the position p and the value v of first different element in $\mathcal{O}(n^2)$.

How many ways are there to construct a valid suffix of permutation? Let's virtually delete all vertices with already fixed values. The heap brokes into some number of lesser heaps with the restrictions that all vertices in some heap should be bigger than value in its parent.

C. Heap

Let's look at the $\mathcal{O}(n^3)$ solution. We want to find the number of smaller d-ary heaps, so we can fix the position p and the value v of first different element in $\mathcal{O}(n^2)$.

How many ways are there to construct a valid suffix of permutation? Let's virtually delete all vertices with already fixed values. The heap brokes into some number of lesser heaps with the restrictions that all vertices in some heap should be bigger than value in its parent.

Suppose we have k heaps $h_1, h_2, \ldots, h_k$ with sizes $|h_1|, |h_2|, \ldots, |h_k|$ and the restrictions that in i-th heap all numbers should be at least x_i.

C. Heap

Let's look at the $\mathcal{O}(n^3)$ solution. We want to find the number of smaller d-ary heaps, so we can fix the position p and the value v of first different element in $\mathcal{O}(n^2)$.

How many ways are there to construct a valid suffix of permutation? Let's virtually delete all vertices with already fixed values. The heap brokes into some number of lesser heaps with the restrictions that all vertices in some heap should be bigger than value in its parent.

Suppose we have k heaps $h_1, h_2, \ldots, h_k$ with sizes $|h_1|, |h_2|, \ldots, |h_k|$ and the restrictions that in i-th heap all numbers should be at least x_i.

Let $cnt_x^{(p)}$ be a number of elements with indices not less than p that are greater than x (and without the value v).

C. Heap

Let's sort all heaps h_i such that x_i will be a non-increasing sequence. Than the number of ways to build the suffix of permutation is

$$\prod_{i=1}^{k} \binom{cnt_{x_i}^{(p)} - \sum_{j=1}^{i-1} |h_j|}{|h_i|} T(h_i)$$

, where $T(h_i)$ is a number of ways to construct a valid d-ary heap h_i.

C. Heap

Let's sort all heaps h_i such that x_i will be a non-increasing sequence. Than the number of ways to build the suffix of permutation is

$$\prod_{i=1}^{k} \binom{cnt_{x_i}^{(p)} - \sum_{j=1}^{i-1} |h_j|}{|h_i|} T(h_i)$$

, where $T(h_i)$ is a number of ways to construct a valid d-ary heap h_i.

This formula follows from that we need to choose $|h_1|$ elements from the $cnt_{x_1}^{(p)}$ appropiate values for a set of values to first heap, and multiply it by $T(h_1)$. These $|h_1|$ elements can't be used in other heaps, and as $x_1 \geqslant x_i$ for $i \geqslant 1$, we won't be able to use these elements in all heaps, but they are counted in all $cnt_{x_i}^{(p)}$-s.

C. Heap

The number of ways to construct a heap h_i with vertices $j = 1 \ldots |h_i|$ and sizes of subtrees $size_j$ is a

$$T(h_i) = \frac{|h_i|!}{\prod_{j=1}^{|h_i|} size_j}$$

This formula can be derived from simple combinatoric observations.

C. Heap

The number of ways to construct a heap h_i with vertices $j = 1 \ldots |h_i|$ and sizes of subtrees $size_j$ is a

$$T(h_i) = \frac{|h_i|!}{\prod_{j=1}^{|h_i|} size_j}$$

This formula can be derived from simple combinatoric observations.

This solution works in $\mathcal{O}(n^3)$. To speed up it to $\mathcal{O}(n^2)$ we need to notice that the majority of binomial coefficients don't change between different values of v. The only thing that can happen is that some heap h_i will swap its order with subtrees of p, when v becomes less than x_i. Using this observation we can keep the current product of binomial coefficients and solve the problem in $\mathcal{O}(n^2)$ time.

D. Lines

We are given a n lines on plane, and we should select the maximum possible number of lines such that no two selected lines are parallel, and no two selected lines have an intersection at point with $x = 0$.

D. Lines

First, let's get rid of vertical lines. We can take only one vertical line to the answer, and it may be either any line that is not line $x = 0$, or it can be line $x = 0$ if it is the only line on plane at all.

D. Lines

First, let's get rid of vertical lines. We can take only one vertical line to the answer, and it may be either any line that is not line $x = 0$, or it can be line $x = 0$ if it is the only line on plane at all.

Now, we can represent each line in form $y = kx + b$. Two parallel lines have the same k, and two lines with intersection at point $(0, b)$ have the same b. So, we should take the maximum number of lines with different k-s and b-s.

D. Lines

Let's construct a bipartite graph with vertices corresponding to different k-s in a left part, and with vertices corresponding to different b-s in a right part. For each line we make an edge between corresponding k, b.

D. Lines

Let's construct a bipartite graph with vertices corresponding to different k-s in a left part, and with vertices corresponding to different b-s in a right part. For each line we make an edge between corresponding k, b.

We need to select maximum number of edges such that no two edges share a vertex. It is maximum matching problem in bipartite graph, and we can solve it with Kuhn's algorithm.

Solution with Kuhn's algorithm runs in $\mathcal{O}(nm)$ time.

E. Yet Another Problem About Permutations

In this problem we need to represent a given permutation as a product of minimal number of simple (each cycle has length no more than 2) permutations.

E. Yet Another Problem About Permutations

There are few cases. First, if the permutation is $p_i = i$, then answer is 0. Second, if all cycles in permutation have lenth no more than 2, answer is 1 — this permutation itself.

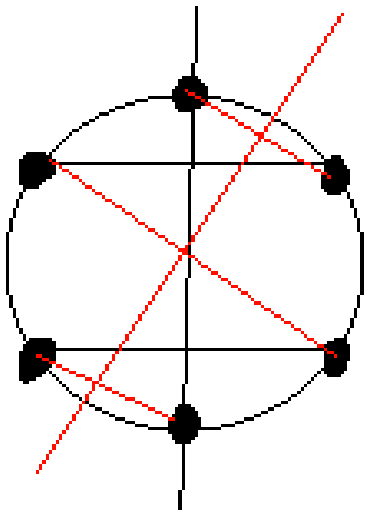
E. Yet Another Problem About Permutations

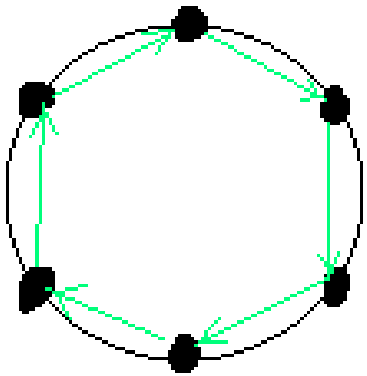
There are few cases. First, if the permutation is $p_i = i$, then answer is 0. Second, if all cycles in permutation have lenth no more than 2, answer is 1 — this permutation itself.

Otherwise, answer is always 2. Let's show this in constructive way: solve the problem independently for all cycles. Suppose we have a cycle with length $l > 2$. We can draw all elements of this cycle on a circle in a vertices of regular polygon. We need to find a way to represent a cyclic shift of these vertices as a composition of two simple permutations.

We can make two symmetry operations with two lines as shown on the next slide.

E. Yet Another Problem About Permutations





First, we make a symmetry with black line, and then symmetry with the red line. We can see, that both symmetries define a simple permutation (every element either remains on its place, or it is swapped with another element), and their composition (first black symmetry, then red symmetry) exactly shifts all vertices on cycle.

F. Strange Sequence

In this problem we need to find a length of n-th term in look-and-say sequence.

F. Strange Sequence

The main idea of the solution is to choose some set of strings such that they don't affect each other.

F. Strange Sequence

The main idea of the solution is to choose some set of strings such that they don't affect each other.

Let $f(s)$ be a result of applying our operation to a string s, and $f(f(f \dots f(s)))$, where there are k iterations of f, be $f^{(k)}(s)$.

Let's look at the example: let $s = 13221133112$, and $f(s) = 11132221232112$. It can be proved that $f(s) = 1113222|12|32112$. In other words, $f^{(k)}(s) = f^{(k)}(13221133112) = f^{(k-1)}(11132221232112) = f^{(k-1)}(1113222) + f^{(k-1)}(12) + f^{(k-1)}(32112)$.

F. Strange Sequence

The main idea of the solution is to choose some set of strings such that they don't affect each other.

Let $f(s)$ be a result of applying our operation to a string s, and $f(f(f \dots f(s)))$, where there are k iterations of f, be $f^{(k)}(s)$.

Let's look at the example: let $s = 13221133112$, and $f(s) = 11132221232112$. It can be proved that $f(s) = 1113222|12|32112$. In other words, $f^{(k)}(s) = f^{(k)}(13221133112) = f^{(k-1)}(11132221232112) = f^{(k-1)}(1113222) + f^{(k-1)}(12) + f^{(k-1)}(32112)$.

We can see that the last symbol of $f^{(k)}(1113222)$ is always 2, and the first symbol of $f^{(k)}(12)$ is always 1 or 3.

F. Strange Sequence

So, we need to find a way to select a finite set of non-affecting strings $\{t_1, t_2, \ldots, t_n\}$ such that $f(t_i) = t_{j_1} t_{j_2} \ldots t_{j_{k_i}}$ for each i.

F. Strange Sequence

So, we need to find a way to select a finite set of non-affecting strings $\{t_1, t_2, \ldots, t_n\}$ such that $f(t_i) = t_{j_1} t_{j_2} \ldots t_{j_{k_i}}$ for each i.

It turns out that it is suffice to generate this set recursively: if we have some string t_i, we can just greedy divide $f(t_i)$ to substrings $a_1 a_2 \ldots a_t$ such that for each j the last symbol of a_j is always different from the first symbol of $f^{(k)}(a_{j+1})$ for each k. It is hard to check this condition for each k, but it is enough to check this condition for $k \leqslant 30$. We can add every string a_j from such partition to our set $\{t_i\}$ and call our partition function recursively for all new substrings.

F. Strange Sequence

After finish of the described algorithm, we get 93 different strings in the set $\{t_i\}$.

Now we can see that $|s_n|$ is a linear recurrence and we can compute n-th element of a recurrence in $\mathcal{O}(|\{t_i\}|^3 \log n)$ by binary exponentiation.

G. Shortest Accepted Word

In this problem we need to construct a lexicographically smallest word accepted by given regular expression.

G. Shortest Accepted Word

We need to build a parse tree of given expression and construct lex. smallest word in each of its vertices:

G. Shortest Accepted Word

We need to build a parse tree of given expression and construct lex. smallest word in each of its vertices:

- from leaf vertex `a`, `b` or `c` we return just corresponding symbol;

G. Shortest Accepted Word

We need to build a parse tree of given expression and construct lex. smallest word in each of its vertices:

- from leaf vertex `a`, `b` or `c` we return just corresponding symbol;
- from iteration vertex $*$ we return "$", because it is the lex.min word;

G. Shortest Accepted Word

We need to build a parse tree of given expression and construct lex. smallest word in each of its vertices:

- from leaf vertex `a`, `b` or `c` we return just corresponding symbol;
- from iteration vertex $*$ we return "$", because it is the lex.min word;
- from concatenation vertex `PQ` we return concatenation of their words;

G. Shortest Accepted Word

We need to build a parse tree of given expression and construct lex. smallest word in each of its vertices:

- from leaf vertex `a`, `b` or `c` we return just corresponding symbol;
- from iteration vertex `*` we return `"$"`, because it is the lex.min word;
- from concatenation vertex `PQ` we return concatenation of their words;
- from union vertex `P|Q` we return the minimum of their words.

We can see that these rules construct the smallest words in each vertex. So, the only problem is to construct a parse tree, which can be done by recursive descent.

H. Work

In this problem we need to assign a set of works to workers, so the distribution is as closest is possible to uniform.

H. Work

Suppose i-th worker is assigned to x_i works. Then we need to minimize $\sqrt{\sum_{i=1}^{n}(x_i - \frac{m}{n})^2}$. It is equivalent to minimize its square: $\sum_{i=1}^{n}(x_i - \frac{m}{n})^2 = \sum_{i=1}^{n} x_i^2 - 2\frac{m}{n}\sum_{i=1}^{n} x_i + n\frac{m^2}{n}$. As each job that can be completed should be completed, we know that $\sum_{i=1}^{n} x_i = const$, so we need only to minimize $\sum_{i=1}^{n} x_i^2$ with given condition on the sum of x_i.

H. Work

This task can be solved by min cost max flow algorithm. Let's construct the following network:

H. Work

This task can be solved by min cost max flow algorithm. Let's construct the following network:

- layer with n vertices, corresponding to workers;

H. Work

This task can be solved by min cost max flow algorithm. Let's construct the following network:

- layer with n vertices, corresponding to workers;
- layer with m vertices, corresponding to works;

H. Work

This task can be solved by min cost max flow algorithm. Let's construct the following network:

- layer with n vertices, corresponding to workers;
- layer with m vertices, corresponding to works;
- edge between worker i and work j with $cap = 1, cost = 0$ if this worker can be assigned to this job;

H. Work

This task can be solved by min cost max flow algorithm. Let's construct the following network:

- layer with n vertices, corresponding to workers;
- layer with m vertices, corresponding to works;
- edge between worker i and work j with $cap = 1, cost = 0$ if this worker can be assigned to this job;
- for each work j edge from its vertex to sink with $cap = 1, cost = 0$;

H. Work

This task can be solved by min cost max flow algorithm. Let's construct the following network:

- layer with n vertices, corresponding to workers;
- layer with m vertices, corresponding to works;
- edge between worker i and work j with $cap = 1, cost = 0$ if this worker can be assigned to this job;
- for each work j edge from its vertex to sink with $cap = 1, cost = 0$;
- for each worker i m edges from sink to its vertex; i-th edge with $cap = 1, cost = 2i - 1$.

H. Work

We can see that min cost max flow in this network is the solution of the task: each doable work will be done, and if worker i executes x_i works it will cost $\sum_{j=1}^{x_i} 2j - 1 = x_i^2$, so between all arrangements our algorithm would choose the one with minimal $\sum_{i=1}^{n} x_i^2$.

The solution works in $\mathcal{O}((n+m)nm)$, if we implement min cost max flow with Dijkstra's algorithm.