ByteDance Finals 2022 Editorial

Jingbang Chen

Georgia Institute of Technology

Acknowledgements

This contest will also be used for Opencup, Grand Prix of ByteDance.

Many thanks to ***ByteDance*** and ***Moscow Workshop*** for hosting this camp and sponsoring this contest setting. Thanks to ***Oleg Hristenko*** for providing contest platform and all necessary technical support. Thanks to ***Emma Yu*** and all other ByteDance members for preparing the camp. Thanks to all coaches (***Yuhao Du, Ruyi Ji, Songyang Chen, Minghong Gao***) and our testing team (***142857, mayaohua2003, gtrhetr***).

Most importantly, thanks to our great problem setting committee: ***chenjb, Claris, Gromah, nothing100, Orenji.Sora, Tommyr7, TsReaper, elfness, shb123, jiangshibiao, oipotato, skywalkert, xyz2606, quailty***. Again, again and yet another again, all of you devotes your professional experience and passionate on this competition, providing us with such a high quality contest. I am sure that all contestants will gain useful knowledge and experience from it.

The committee also feel quite fun to mention a fact here, that the union set of contests that our members set before includes 22 ICPC regional of the EC Region and 10 CCPC contests from 2015 to 2021. We will provide a list here just in case anyone is interested:

ICPC:

- 2015: Shanghai
- 2018: Qingdao, Xuzhou, Shenyang, Jiaozuo, EC Final
- 2019: Nanjing, Yinchuan, Nanchang, Hong Kong, EC Final
- 2020: Nanjing, Shenyang, Shanghai, Yinchuan, Macau, EC Final
- 2021: Nanjing, Shenyang, Shanghai, Macau, EC Final

CCPC:

- 2016: Hangzhou
- 2017: Hangzhou, Qinhuangdao
- 2018: Guilin
- 2019: Harbin, Qinhuangdao
- 2020: Mianyang, Changchun
- 2021: Harbin, Guilin

Wish everyone best luck in the future!

A. Driverless Car II

Author: quailty

If there exist two transmitters that have the same distance to the car, the area of such positions is zero. We can safely assume that the car has pairwise distinct distances to all the transmitters.

Now we can enumerate the closest transmitter and the second closest transmitter to the car. To satisfy the conditions, the car must located in the intersection of an ellipse and $O(n)$ half-planes. To calculate the area of the intersection, we first apply the $O(n \log n)$ time half-plane intersection algorithm on a sufficiently large bounding box and these half-planes to get a (convex) polygon, then we apply coordinates dilation along the major axis of the ellipse to make it a circle, and we finally need to find the intersection of a polygon and a circle, which is a typical computation geometry problem.

However, simply enumerating all pairs of transmitters results in an $O(n^3 \log n)$ time complexity, which is insufficient to fit in the time limit. To speed this up, we can build a Voronoi diagram of the transmitters. When we enumerate the closest transmitter to the car, the second closest transmitter must be in the neighbouring Voronoi cell of the cell that the closest transmitter be in. It can be shown that the total number of pairs enumerated is $O(n)$, and the time complexity is $O(n^2 \log n)$ in all.

B. Longest Increasing Subsequence

Author: elfness

Firstly we can split the final sequence into several levels, where each level is combination of at most $n-1$ subsequences:

- Base : a_1
- Level 0 : a_2, $\cdots$,a_{n-1}, a_n
- Level 1 : $S_{1,1}$, $S_{2,1}$, $\cdots$, $S_{n-1,1}$
- $\cdots$
- Level m : $S_{1,m}$, $S_{2,m}$, $\cdots$, $S_{n-1,m}$

As for the example input $\{1, 5, 20\}$, the splited levels will be:

- Base : 1
- Level 0 : 5, 20
- Level 1 : $\{3\}$, $\{12\}$
- Level 2 : $\{2, 4\}$, $\{8, 16\}$
- Level 3 : $\{\}$, $\{6, 10, 14, 18\}$
- Level 4 : $\{\}$, $\{7, 9, 11, 13, 15, 17, 19\}$

Here we have $\sum_{j=1}^{m} |S_{i,j}| = a_{i+1} - a_i = D_i$, and we can prove that $S_{i,1} = 1$, $S_{i,2} = 2$, $\cdots$, $S_{i,last_i} = D_i - 2^{last_i - 1} + 1$, $S_{i,j} = 0\,(j > last_i)$. And approximately, it's not worthy to split a subsequence $S_{i,j}$ if we want to get the longest increasing subsequence.

So we can use dynamic programming to find LIS, use $f_{i,j}$ to represent the length of LIS ending with subsequence $S_{i,j}$, then $f_{i,j} = max\{f_{i-1,k} \,|\, k \le j\} + L_{i,k,j}$, where $L_{i,k,j}$ represents the length of LIS of $\{S_{i,k}, S_{i,k+1}, ..., S_{i,j}\}$. Here $L_{i,k,j} = |S_{i,j}|$ for $j < last_i$, but for $j \ge last_i$, $S_{i,last_i - 1}$ and $S_{i,last_i}$ may combine to get a longer subsequence.

Time complexity: $O(n \log C)$.

C. New Equipments III

Author: Claris

This is probably a typical minimum cost problem on a network with $O(n)$ vertices and $O(m)$ arcs, if we negate all the values in the matrix p, build a weighted bipartite graph that corresponds to the matrix p where all the zero-weighed edges are omitted, and build a network based on the bipartite graph.

However, applying inefficient algorithm, e.g. the successive shortest path algorithm in $O(nm \log m)$ time by augmenting the flow along one shortest path in residual network with reduced costs step by step, can hardly fit in the time limit. The indented solution runs in $O(tm\sqrt{n})$ time, where $t = 5$ is the maximum possible value in the matrix p, based on the primal-dual algorithm.

The main idea is that, instead of augmenting the flow along one shortest path, in each step we use Dinic's algorithm to find the maximum flow through the so called admissible network, which contains only those arcs with a zero reduced cost, and then the length of the shortest augment path increases strictly. Since the length of the shortest augment path is always a negative integer no less than t, it needs to run Dinic's algorithm on a unit network at most t times.

You may check MINIMUM COST FLOW PART TWO: ALGORITHMS for more information on the the primal-dual algorithm.

D. Interesting String Problem

Author: oipotato

Consider building suffix trees for the string. Obviously, all substrings represented by a node in the suffix tree must be continuous in the sequence. Calculate the prefix sum of the lengths by dfs order and then you can find the node of the query by binary search. In order to find the position of the character, you should solve such problem that find the k-th number in a subtree. This is a classic problem, you can solve it by persistent segment tree.

E. Card Shark

Author: TsReaper

Let's rephrase the problem in a more formal way:

Given n strings consisting of 0s and 1s, please concat all strings into a big string $s_0 s_1 \cdots s_{k-1}$ so that for all $0 \le i < k$, if $i \ mod \ m = b$ then $s_i = 1$, otherwise $s_i = 0$.

Consider a given string $t_0 t_1 \cdots t_{l-1}$. As there is at least one 1 in each string, let $t_j = 1$ and let t_0 be the i-th (indices starts from 0) character in the big string, we have $i + j \equiv b \pmod{m}$, so $i \equiv b - j \pmod{m}$.

Let's construct a graph with m vertices. For string $t_0 t_1 \cdots t_{l-1}$ we connect an edge from vertex $(b - j) \mod m$ to $(b - j + l) \mod m$. As the length of the big string is divisible by m, if we can find a Euler circuit starting from vertex 0 and also ending at vertex 0 then we have an answer.

What's left is to find the answer with the smallest lexicographic order. We just need to follow the edge with the smallest index each time in DFS when calculating the Euler circuit.

F. Coprime Matrices

Author: Gromah

Firstly ignore the constraint $M_{x,y} = w$, we can divide the matrix into several $n \times 2$ strips, and probably a single chain (if m is odd), then fill each region($n \times 2$ strip or $n \times 1$ chain) in a snake-like way. Here is a example where $n = 5, m = 5$:

1	2	11	12	21
4	3	14	13	22
5	6	15	16	23
8	7	18	17	24
9	10	19	20	25

As we can see, each vertical internal number x is adjacent to $x - 1$ or $x + 1$, which is coprime with x, and so do horizontal internal numbers.

Now consider the constraint $M_{x,y} = w$, we need just shift the numbers to make $M_{x,y} = w$ since $\gcd(1, nm) = 1$ so that the property above still holds.

G. Factor

Author: elfness

Firstly we need to find out whether a number x is a good integer. if $x = \prod_{i=1}^{m} p_i^{a_i} (p_i < p_{i+1})$, let $S_i = \prod_{j=1}^{i} (\sum_{k=0}^{a_j} p_j^k), S_0 = 1$, x is a good integer if and only if for all $1 \le i \le m$, $p_i \le S_{i-1} + 1$.

$Proof$:

- $Necessity$: if for some i, $p_i > S_{i-1} + 1$, then there is no subset with $sum = p_i - 1$, because the sum of factors of n whose prime factors are all less than p_i exactly equals S_{i-1} and $S_{i-1} < p_i - 1$, and other factors are all $\ge p_i$, which are useless to make up $p_i - 1$.
- $Sufficiency$: proved by induction, assume that we can find a subset whose sum equals y for all integer $1 \le y \le S_{i-1}$, as for each integer $p_i \le z \le S_i$, we can write $z = \sum_{j=0}^{a_i} b_j {p_i}^j (0 \le b_j \le S_{i-1})$, then we can find the subset by construction: for each integer $0 \le j \le a_i$, find the subset whose sum equals b_j, then multiply all integers in this subset by ${p_i}^j$. Union the $a_i + 1$ subsets we can get a big subset whose sum equals z. So for all integers $1 \le y \le S_i$ we can find a subset whose sum equals y.

Then we can count the number of good integers by depth-first-search on prime numbers, for a good integer x there are two situations:

- $a_m = 1$: we can determine the number of them by enumerating $C = \prod_{i=1}^{m-1} p_i^{a_i}$, where for all integers $1 \le j < m$, $p_j \le S_{j-1} + 1$, and count the number of valid p_m's. So we can use depth-first-search on prime numbers to enumerate all valid C's, then a prime number p_m is

valid iff $p_m > p_{m-1}$ and $p_m \leq S_{m-1} + 1$ and $C \times p_m \leq n$, here $C \times p_m$ is a good integer in this situation. We can use prefix-sum to calculate how many prime numbers are there in the range $\left[p_{m-1}+1, min\{S_{m-1}+1, \lfloor \frac{n}{C} \rfloor\}\right]$, where one upper bound of p_m may be 2035002 when $C = 491400 = 2^3 \times 3^3 \times 5^2 \times 7 \times 13, S_5 = 2083200, \lfloor \frac{10^{12}}{C} \rfloor = 2035002$.
- $a_m > 1$: we can simply use depth-first-search to enumerate all of them.

When $n = 10^{12}$, the total number of search iterations will be about 10^8, which costs about 2 seconds.

H. Graph Operation

Author: Tommyr7

Obviously, no matter how we operate, the degree of each vertex doesn't change. Therefore, graph G can be changed to graph H only when degrees of corresponding vertices in two graphs are the same. Now, let's prove that the condition is strong enough.

Let's focus on a single vertex u. Define A as the set of vertices that are adjacent to u in graph G, and B as the set of vertices that are adjacent to u in graph H.

If $A = B$, then we just ignore the vertex u and consider the remaining part.

If $A \neq B$, then there exists two different vertices v and w such that $u \sim v, u \not\sim w$ in graph G and $u \not\sim v, u \sim w$ in graph H, there are two situations we could check:

- There exists another vertex t such that $v \not\sim t, w \sim t$ in graph G. We just select u, v, w, t and perform an operation on graph G. Then, we repeat the process.
- There exists another vertex t such that $v \sim t, w \not\sim t$ in graph H. We just select u, w, v, t and perform an operation on graph H. Then, we repeat the process.

If the first situation doesn't occur, we can know that $deg_v > deg_w$, and if the second situation doesn't occur, we can know that $deg_w > deg_v$. Since the degrees of v and w in graph G are equal to those in graph H, at least one of the situations will occur.

Finally, we just need to output the operations we have performed on graph G, and then output the operations we have performed on graph H in reverse. Using bitset to speed up the process of finding vertex t, the time complexity is $O(\frac{n^3}{\omega})$.

I. Optimal Assortment

Author: nothing100

For any given toy set S, the customer's preference in the worst case is $w_0 = r_0$ and $w_i = l_i$ for all $i > 0$. So the problem becomes to get the optimal set under such preference. And we can use binary search to get the optimal set since we can check whether the answer is greater than p by adding all the toys with $v > p$ into the set and checking the profit greater than p or not.

However, there are series of modify operations, so we need to use some data structures such as segment tree, self-balancing binary tree to determine the answer quickly after each operation. $O(m \log n)$ or quick $O(m \log^2 n)$ time complexity could be acceptable.

J. Cell Tower

Author: jiangshibiao

8×8 seems to be too large for using state compressed dynamic programming to solve directly. Usually under such circumstance, we may guess that the number of useful states might be limited in an acceptable range. We define $f(S)$ as the possible ways for filling the current grids to S, and we use hash table and queue to maintain all useful states. We precompute all patterns of size 3 and 4. Then for each transition, we enumerate all patterns to fill the empty grid with lowest index. Therefore, we can guarantee that no repetition nor missing in our brute-force. To accelerate the transition, we can also precompute all possible patterns for each grid. Therefore, we can guarantee that all enumeration is a legal transition. The time complexity is equal to the number of legal transitions. For $n = 11000$, S and the number of legal transitions are at 10^6 leval, which is sufficient to pass this problem in our time limit.

K. Xiangqi

Author: shb123

We can observe that the cannon can move to any position in at most 4 steps. Therefore, unless the horse is very near to the king, using the cannon to attack the king is a better choice, because moving to the nearest axis is faster than moving to the original point for the horse.

The case we use the horse to attack the king only happens when the initial position of the horse is near to the king, for example, in the range $[-15, 15] \times [-15, 15]$. We can use BFS to solve this case.

The second case is that we use the cannon to attack. Because of the symmetry, we can always assume that $x_h \geq 0, y_h \geq 0$, and the cannon attacks the king from the x-axis. We solve the positive half-axis and the negative half-axis separately. If the cannon has been on the x-axis, we only need one extra step to move the cannon to $(\pm\infty, 0)$. Therefore, we only need to solve one problem: What is the minimum number of steps needed to move the horse to the corresponding half-axis? After minimizing the number of steps, we need to minimize the distance between the final position of the horse and the king.

For the positive half-axis, the minimum number of steps is always $\lceil \frac{y_h}{2} \rceil$. We just calculate all the possible final positions and find the minimum positive one.

For the negative half-axis, we first try to move to the x-axis with $\lceil \frac{y_h}{2} \rceil$ steps and do a similar adjustment. In the case that the leftmost position we can achieve in this way is still on the positive half-axis, we can try to replace an $(-1, -2)$ in the operation sequence with two $(-2, -1)$, that move the final position 3 units left with one extra step. If it's still not enough, we can use $(-2, 1)+(-2, -1)$ to move the final position 4 units left with two steps. Notice that it's better to jump from $(1, 0)$ to $(-1, 0)$ with operation $(-1, 2) + (-1, -2)$(Although it doesn't affect the final answer).

A special case: the horse is at $(100, 0)$ and the cannon is at $(99, 0)$. We need 3 steps to move the cannon to $(+\infty, 0)$, and it's better to move the horse. There might be some other ignored details, try to find them by yourselves.