ACM ICPC Asia-Amritapuri Site Onsite round 2018

Problems editorial

Gleb Evstropov

December 27th, 2017

Nested Candy Boxes

Every box of level one contains a_1 candies, every box of level two contains a_2 boxes of level one, every box of level three contains a_3 boxes of level two and so on. What is the minimum number of boxes you have to open in order to get x candies? Process m different values of x.

Nested Candy Boxes

If we have two unopened boxes of different levels $i > j$. Any optimal strategy will open box of level j first.

Nested Candy Boxes

If we have two unopened boxes of different levels $i > j$. Any optimal strategy will open box of level j first.

In terms of a tree it means we visit all the subtree before leaving the node.

Nested Candy Boxes

First we describe a way to process one query in $O(n)$ time.

Nested Candy Boxes

First we describe a way to process one query in $O(n)$ time.

For each box level we compute the number of candies inside it. Let this value be c_i.

Nested Candy Boxes

First we describe a way to process one query in $O(n)$ time.

For each box level we compute the number of candies inside it. Let this value be c_i.

Replace it with ∞ if $c_i \geqslant 10^{18}$.

Nested Candy Boxes

Now, for each level i we have to open $\lceil \frac{x}{c_i} \rceil$ boxes.

Nested Candy Boxes

Now, for each level i we have to open $\lceil \frac{x}{c_i} \rceil$ boxes.

To process one query in $O(\log x)$ time we notices that there are no more than $\log x$ different values of c_i (after we replace too large values with ∞).

Longest Races

You are given a value k, construct a tree that has exactly k diameters. The number of vertices used should be minimum possible.

Longest Races

We should consider four different cases of diameter length.

Longest Races

We should consider four different cases of diameter length.

If $d = 1$, the tree consists of two vertices and one edge, that is the optimal solutions for $k = 1$.

Longest Races

We should consider four different cases of diameter length.

If $d = 1$, the tree consists of two vertices and one edge, that is the optimal solutions for $k = 1$.

If $d = 2$, the graph if a star. That would be an optimal solution for $k = \frac{x(x-1)}{2}$.

Longest Races

We should consider four different cases of diameter length.

If $d = 1$, the tree consists of two vertices and one edge, that is the optimal solutions for $k = 1$.

If $d = 2$, the graph if a star. That would be an optimal solution for $k = \frac{x(x-1)}{2}$.

If $d = 3$, the tree is an edge with a star-graph on each of its endpoints. If one of them has size a and the other has size b, the total number of diameters will be $a \cdot b$. In particular, for any value of k there exists a solutions $a = 1$, $b = k$.

Longest Races

Finally, $d = 4$ is the general case. A single node has a set of brooms (start plus one edge) hanging on it. If the brooms have $a_1, a_2, \ldots, a_l$ leaves, the number of diameters is $a_1 \cdot a_2 + (a_1 + a_2) \cdot a_3 + \ldots + (a_1 + a_2 + \ldots + a_{l-1}) \cdot a_l$.

Longest Races

Finally, $d = 4$ is the general case. A single node has a set of brooms (start plus one edge) hanging on it. If the brooms have $a_1, a_2, \ldots, a_l$ leaves, the number of diameters is $a_1 \cdot a_2 + (a_1 + a_2) \cdot a_3 + \ldots + (a_1 + a_2 + \ldots + a_{l-1}) \cdot a_l$.

Compute dynamic programming $d(i, j)$ — what is the minimum number of nodes required to get tree with i leaves and j diameters. Try every new size of the broom x to add. This can be computed in $O(k^3)$ time.

Longest Races

Finally, $d = 4$ is the general case. A single node has a set of brooms (start plus one edge) hanging on it. If the brooms have $a_1, a_2, \ldots, a_l$ leaves, the number of diameters is $a_1 \cdot a_2 + (a_1 + a_2) \cdot a_3 + \ldots + (a_1 + a_2 + \ldots + a_{l-1}) \cdot a_l$.

Compute dynamic programming $d(i, j)$ — what is the minimum number of nodes required to get tree with i leaves and j diameters. Try every new size of the broom x to add. This can be computed in $O(k^3)$ time.

Pick the solution as the optimum among these four cases.

Magic Board

Given a table of digits you are allowed to read integer moving left, right, up or down. What is the minimum integer you won't be able to read with this procedure?

Magic Board

First idea: the answer is small.

Magic Board

First idea: the answer is small.

Indeed, fix the lenght of the answer k. There are 10^k integers of length $\leqslant k$ and $n \cdot m \cdot 4^{k-1}$ paths of such length.

Magic Board

First idea: the answer is small.

Indeed, fix the lenght of the answer k. There are 10^k integers of length $\leqslant k$ and $n \cdot m \cdot 4^{k-1}$ paths of such length.

Thus, for $k \geqslant log_{\frac{5}{2}}(n \cdot m)$ there will be an integer that we can't read.

Magic Board

First idea: the answer is small.

Indeed, fix the lenght of the answer k. There are 10^k integers of length $\leqslant k$ and $n \cdot m \cdot 4^{k-1}$ paths of such length.

Thus, for $k \geqslant log_{\frac{5}{2}}(n \cdot m)$ there will be an integer that we can't read.

The compexity is $O(n \cdot m \cdot 4^k) = O(n \cdot m \cdot 4^{log_{\frac{5}{2}} n \cdot m}) = O(n \cdot m \cdot (nm)^{log_{\frac{5}{2}} 4})$ that is approximately $O((nm)^{2.5})$.

Magic Board

This problem required no special optimization's and fits in time limit even for $k = 8$ (though, pretty tight).

Magic Board

This problem required no special optimization's and fits in time limit even for $k = 8$ (though, pretty tight).

However, in practice you should increment k one by one until you find the answer. Under the given constrains $k = 6$ would be more than enough.

Drunk Man in Large City

You are given undirected graph, starting vertex s and target vertex t. In one turn you move a chip along any edge, then it arbitrary moves along any edge that starts from the current vertex with except of the edge that was just used. Can you guarantee to reach t, and if so, what is the minimum number of steps required to do this in the worst case?

Drunk Man in Large City

For each vertex of the graph we would like to compute two values:

- $a(v)$ — what is the answer if the first player currently moves from vertex v. There are n such states.
- $b(v, e)$ — what is the answer if the second player currently moves from vertex v and the last edge used was e. There are $2m$ such states.

Drunk Man in Large City

For each vertex of the graph we would like to compute two values:

- $a(v)$ — what is the answer if the first player currently moves from vertex v. There are n such states.
- $b(v, e)$ — what is the answer if the second player currently moves from vertex v and the last edge used was e. There are $2m$ such states.

Assume all the values are known except for $a(v)$ and $b(v)$ for some fixed vertex v, $a(v) = -1$ (first player looses) if $b(u, e) = -1$ is true for all $e \in N(v)$. Otherwise $a(v) = min(b(u, e))$ for all $e \in N(v)$ such that $b(u, e) \neq -1$.

Drunk Man in Large City

For each vertex of the graph we would like to compute two values:

- $a(v)$ — what is the answer if the first player currently moves from vertex v. There are n such states.
- $b(v, e)$ — what is the answer if the second player currently moves from vertex v and the last edge used was e. There are $2m$ such states.

Assume all the values are known except for $a(v)$ and $b(v)$ for some fixed vertex v, $a(v) = -1$ (first player looses) if $b(u, e) = -1$ is true for all $e \in N(v)$. Otherwise $a(v) = min(b(u, e))$ for all $e \in N(v)$ such that $b(u, e) \neq -1$.

Drunk Man in Large City

Similarly $b(v, e) = -1$ if $a(u) = -1$ for at least one $u \in N(v)$, such that $(u, v) \neq e$, or $b(v, e) = max(a(u))$ among all $u \in N(v)$, such that $(u, v) \neq e$.

Drunk Man in Large City

Similarly $b(v, e) = -1$ if $a(u) = -1$ for at least one $u \in N(v)$, such that $(u, v) \neq e$, or $b(v, e) = max(a(u))$ among all $u \in N(v)$, such that $(u, v) \neq e$.

Applying any possible relaxations while at least one of them happens gives us polynomial time solution.

Drunk Man in Large City

Similarly $b(v, e) = -1$ if $a(u) = -1$ for at least one $u \in N(v)$, such that $(u, v) \neq e$, or $b(v, e) = max(a(u))$ among all $u \in N(v)$, such that $(u, v) \neq e$.

Applying any possible relaxations while at least one of them happens gives us polynomial time solution.

We apply a sort of retroanalysis technique. Every time we pick the minimum value and apply relaxations.

Drunk Man in Large City

Similarly $b(v, e) = -1$ if $a(u) = -1$ for at least one $u \in N(v)$, such that $(u, v) \neq e$, or $b(v, e) = max(a(u))$ among all $u \in N(v)$, such that $(u, v) \neq e$.

Applying any possible relaxations while at least one of them happens gives us polynomial time solution.

We apply a sort of retroanalysis technique. Every time we pick the minimum value and apply relaxations.

Value of $a(v)$ is defined as soon as at least one valid $b(u, e)$, $u \in N(v)$ is computed.

Drunk Man in Large City

Similarly $b(v, e) = -1$ if $a(u) = -1$ for at least one $u \in N(v)$, such that $(u, v) \neq e$, or $b(v, e) = max(a(u))$ among all $u \in N(v)$, such that $(u, v) \neq e$.

Applying any possible relaxations while at least one of them happens gives us polynomial time solution.

We apply a sort of retroanalysis technique. Every time we pick the minimum value and apply relaxations.

Value of $a(v)$ is defined as soon as at least one valid $b(u, e)$, $u \in N(v)$ is computed.

Value of $b(v, e)$ is known only if all $a(u)$ are known for all neighbours of v except for other endpoint of e. To compute this case fast we keep track on how many neighbours are not yet computed. When counter goes down to 1 we define value for one edge, when it goes down to 0 we compute it for all the other.

Sliding Puzzle

You are given n rectangles at the plane with sides parallel to coordinate axes. In one unit of time you can pick one rectangle and move it left, right, up or down. What is the minimum time required to make the set of rectangles nested.

Sliding Puzzle

First we should check whether the answer exists. Sort all rectangles by their area, for any two neighbouring check whether one fits inside the other. This can be done in $O(n \log n)$ time.

Sliding Puzzle

First we should check whether the answer exists. Sort all rectangles by their area, for any two neighbouring check whether one fits inside the other. This can be done in $O(n \log n)$ time.

Notice that problem can be solved independently for each of the dimensions. Thus, the problem is, what is the minimum cost required to move all segments in order to make a nested family of segments.

Sliding Puzzle

First we should check whether the answer exists. Sort all rectangles by their area, for any two neighbouring check whether one fits inside the other. This can be done in $O(n \log n)$ time.

Notice that problem can be solved independently for each of the dimensions. Thus, the problem is, what is the minimum cost required to move all segments in order to make a nested family of segments.

Sort all segments in order of ascending lengths. We would like to compute the following function: $f(i, x)$ — what is the minimum cost required to make first i segments located in valid order with the i-th largest starting at point x.

Sliding Puzzle

$f(i,x) = |x - s_i| + \min f(i-1, y)$ for $y \in [x; x + len_i - len_{i-1}]$. Here len_i stands for the length of the i-th segment and s_i is its initial left endpoint coordinate.

Sliding Puzzle

$f(i, x) = |x - s_i| + \min f(i-1, y)$ for $y \in [x; x + len_i - len_{i-1}]$. Here len_i stands for the length of the i-th segment and s_i is its initial left endpoint coordinate.

The result is always a piecewise linear function. For each block we keep track of its x-length and tangent. Moreover, we know that this function is convex downward (i.e. sequence of tangents is non-decreasing). Consider operations we should apply:

Sliding Puzzle

$f(i, x) = |x - s_i| + \min f(i-1, y)$ for $y \in [x; x + len_i - len_{i-1}]$. Here len_i stands for the length of the i-th segment and s_i is its initial left endpoint coordinate.

The result is always a piecewise linear function. For each block we keep track of its x-length and tangent. Moreover, we know that this function is convex downward (i.e. sequence of tangents is non-decreasing). Consider operations we should apply:

- $\min f(i-1, y)$ for $y \in [x; x + y]$ is equal to extending the lowest segment by y.

Sliding Puzzle

$f(i, x) = |x - s_i| + \min f(i-1, y)$ for $y \in [x; x + len_i - len_{i-1}]$. Here len_i stands for the length of the i-th segment and s_i is its initial left endpoint coordinate.

The result is always a piecewise linear function. For each block we keep track of its x-length and tangent. Moreover, we know that this function is convex downward (i.e. sequence of tangents is non-decreasing). Consider operations we should apply:

- $\min f(i-1, y)$ for $y \in [x; x + y]$ is equal to extending the lowest segment by y.
- Adding linear function $|x - s_i|$ splits one segment in two and changes all tangents by 1.

Sliding Puzzle

This can be implemented by storing the whole function in the array. For each segment we store its starting point and its tangent. Recomputing takes $O(m)$ where m is the number of segments that don't exceed n, so the total running time is $O(n^2)$.

Sliding Puzzle

This can be implemented by storing the whole function in the array. For each segment we store its starting point and its tangent. Recomputing takes $O(m)$ where m is the number of segments that don't exceed n, so the total running time is $O(n^2)$.

To make the algorithm work in $O(n \log n)$ bruteforce power of balanced BST can be used. However, we can deal using only standard stl set. We split the function at its lowest point and keep all positions of a tangent change in a set. Also we keep in a separate variable the overall shift of all value in the second set. Note that we might need to store the same x coordinate twice or more times in the same set (if the function changes by more than 1).

Sliding Puzzle

This can be implemented by storing the whole function in the array. For each segment we store its starting point and its tangent. Recomputing takes $O(m)$ where m is the number of segments that don't exceed n, so the total running time is $O(n^2)$.

To make the algorithm work in $O(n \log n)$ bruteforce power of balanced BST can be used. However, we can deal using only standard stl set. We split the function at its lowest point and keep all positions of a tangent change in a set. Also we keep in a separate variable the overall shift of all value in the second set. Note that we might need to store the same x coordinate twice or more times in the same set (if the function changes by more than 1).

To add the new value we inset the coordinate in a set and may be move one element between two sets. To apply minimum we simple increase the shift of the second set by the corresponding value.