9th round contest

August 14, 2022

Cup Ukainian of Programming

A. ACTG Matrix

Find substrings of given strings s and t with the largest *similarity*.

A. ACTG Matrix

Define $match(c_1, c_2) = \begin{cases} 1, & c_1 = c_2 \\ 0, & c_1 \neq c_2 \end{cases}$.

A. ACTG Matrix

Define $match(c_1, c_2) = \begin{cases} 1, & c_1 = c_2 \\ 0, & c_1 \neq c_2 \end{cases}$.

We use DP. $dp_{i,j}$ — the largest similarity between a substring of s ending at index i, and a substring of j ending at index j.

A. ACTG Matrix

Define $match(c_1, c_2) = \begin{cases} 1, & c_1 = c_2 \\ 0, & c_1 \neq c_2 \end{cases}$.

We use DP. $dp_{i,j}$ — the largest similarity between a substring of s ending at index i, and a substring of j ending at index j.

$dp_{i,j}$ is the maximum of the following:

- $max(0, dp_{i-1,j-1}) + match(s_i, t_j)$;

A. ACTG Matrix

Define $match(c_1, c_2) = \begin{cases} 1, & c_1 = c_2 \\ 0, & c_1 \neq c_2 \end{cases}$.

We use DP. $dp_{i,j}$ — the largest similarity between a substring of s ending at index i, and a substring of j ending at index j.

$dp_{i,j}$ is the maximum of the following:

- $max(0, dp_{i-1,j-1}) + match(s_i, t_j)$;
- $dp_{i-1,j} - 1$, $dp_{i,j-1} - 1$.

A. ACTG Matrix

Define $match(c_1, c_2) = \begin{cases} 1, & c_1 = c_2 \\ 0, & c_1 \neq c_2 \end{cases}$.

We use DP. $dp_{i,j}$ — the largest similarity between a substring of s ending at index i, and a substring of j ending at index j.

$dp_{i,j}$ is the maximum of the following:

- $max(0, dp_{i-1,j-1}) + match(s_i, t_j)$;
- $dp_{i-1,j} - 1$, $dp_{i,j-1} - 1$.

The answer is $max_{i,j} dp_{i,j}$. Complexity is $O(|s| \cdot |t|)$.

B. Boundary String

Given is a description of a proper rectilinear polygon boundary, as a sequence of left and right turns. Find the smallest bounding box area.

It is guaranteed the intersection of any vertical line with the polygon interior is a segment (or empty).

B. Boundary String

The restriction tells us about the structure of the polygon: the boundary can be split into the *lower* and *upper halves*, both monotonic in x-coordinate (always go right and up/down).

B. Boundary String

The restriction tells us about the structure of the polygon: the boundary can be split into the *lower* and *upper halves*, both monotonic in x-coordinate (always go right and up/down).

Let us represent each half as a left-to-right sequence of their horizonal sides. Polygons matching the description can be obtained by choosing both *width (horizontal span)* and *height (y-coordinate)* of each horizontal side, so that:

B. Boundary String

The restriction tells us about the structure of the polygon: the boundary can be split into the *lower* and *upper halves*, both monotonic in x-coordinate (always go right and up/down).

Let us represent each half as a left-to-right sequence of their horizonal sides. Polygons matching the description can be obtained by choosing both *width (horizontal span)* and *height (y-coordinate)* of each horizontal side, so that:

- The total widths of both boundary halfs are equal.

B. Boundary String

The restriction tells us about the structure of the polygon: the boundary can be split into the *lower* and *upper halves*, both monotonic in x-coordinate (always go right and up/down).

Let us represent each half as a left-to-right sequence of their horizonal sides. Polygons matching the description can be obtained by choosing both *width (horizontal span)* and *height (y-coordinate)* of each horizontal side, so that:

- The total widths of both boundary halfs are equal.
- At any x-coordinate the upper half is strictly higher than the lower half.

B. Boundary String

The restriction tells us about the structure of the polygon: the boundary can be split into the *lower* and *upper halves*, both monotonic in x-coordinate (always go right and up/down).

Let us represent each half as a left-to-right sequence of their horizonal sides. Polygons matching the description can be obtained by choosing both *width (horizontal span)* and *height (y-coordinate)* of each horizontal side, so that:

- The total widths of both boundary halfs are equal.
- At any x-coordinate the upper half is strictly higher than the lower half.
- For any pair of consecutive horizontal sides in a boundary half, one should be strictly higher/lower than the other depending on the turn directions at those sides.

B. Boundary String

Suppose that the height H of the bounding box is fixed. It is best to place the lower half as low as possible, and the upper half as high as possible, while respecting restrictions on adjacent sides.

B. Boundary String

Suppose that the height H of the bounding box is fixed. It is best to place the lower half as low as possible, and the upper half as high as possible, while respecting restrictions on adjacent sides.

This defines the heights of all sides unambiguously. We only need to determine the smallest total width.

B. Boundary String

Suppose that the height H of the bounding box is fixed. It is best to place the lower half as low as possible, and the upper half as high as possible, while respecting restrictions on adjacent sides.

This defines the heights of all sides unambiguously. We only need to determine the smallest total width.

This can be done with DP. $dp_{i,j}$ = the smallest width of a polygon containing i and j leftmost sides from the lower/upper half respectively.

B. Boundary String

Suppose that the height H of the bounding box is fixed. It is best to place the lower half as low as possible, and the upper half as high as possible, while respecting restrictions on adjacent sides.

This defines the heights of all sides unambiguously. We only need to determine the smallest total width.

This can be done with DP. $dp_{i,j} =$ the smallest width of a polygon containing i and j leftmost sides from the lower/upper half respectively.

Transitions are to either include a new side in one half, or new sides in both. Make sure the halfs don't intersect.

B. Boundary String

Suppose that the height H of the bounding box is fixed. It is best to place the lower half as low as possible, and the upper half as high as possible, while respecting restrictions on adjacent sides.

This defines the heights of all sides unambiguously. We only need to determine the smallest total width.

This can be done with DP. $dp_{i,j} =$ the smallest width of a polygon containing i and j leftmost sides from the lower/upper half respectively.

Transitions are to either include a new side in one half, or new sides in both. Make sure the halfs don't intersect.

Since DP is $O(n^2)$, and $H = O(n)$, this is an $O(n^3)$ solution.

C. Convex Shell

For a convex polyhedron P, find the volume of the set P_d of points at distance at most d from the polyhedron.

C. Convex Shell

Each point $p \in P_d$ we put in one of four possible groups based on where the closest point $q \in P$ is:

C. Convex Shell

Each point $p \in P_d$ we put in one of four possible groups based on where the closest point $q \in P$ is:

- q is strictly inside the polygon (when p also is);

C. Convex Shell

Each point $p \in P_d$ we put in one of four possible groups based on where the closest point $q \in P$ is:

- q is strictly inside the polygon (when p also is);
- q is strictly inside a polygon face;

C. Convex Shell

Each point $p \in P_d$ we put in one of four possible groups based on where the closest point $q \in P$ is:

- q is strictly inside the polygon (when p also is);
- q is strictly inside a polygon face;
- q is strictly inside a polygon edge;

C. Convex Shell

Each point $p \in P_d$ we put in one of four possible groups based on where the closest point $q \in P$ is:

- q is strictly inside the polygon (when p also is);
- q is strictly inside a polygon face;
- q is strictly inside a polygon edge;
- q is a polygon vertex.

C. Convex Shell

For each group we compute the volume:

- q is strictly inside the polygon (when p also is).
  The volume is equal to the volume of P.

C. Convex Shell

For each group we compute the volume:

- q is strictly inside the polygon (when p also is).
  The volume is equal to the volume of P.
- q is strictly inside a polygon face.
  A face of area S contributes $S \cdot d$ to the answer.

C. Convex Shell

For each group we compute the volume:

- q is strictly inside the polygon (when p also is).
  The volume is equal to the volume of P.
- q is strictly inside a polygon face.
  A face of area S contributes $S \cdot d$ to the answer.
- q is strictly inside a polygon edge.
  Each edge contributes $l \cdot d^2 \cdot (\pi - \alpha)$, where l is the edge length, α is the *dihedral angle* between faces adjacent to the edge.

C. Convex Shell

For each group we compute the volume:

- q is strictly inside the polygon (when p also is).
  The volume is equal to the volume of P.
- q is strictly inside a polygon face.
  A face of area S contributes $S \cdot d$ to the answer.
- q is strictly inside a polygon edge.
  Each edge contributes $l \cdot d^2 \cdot (\pi - \alpha)$, where l is the edge length, α is the *dihedral angle* between faces adjacent to the edge.
- q is a polygon vertex.
  For each vertex, the region is a ball wedge. These wedges can be combined to form a single ball of radius d, thus the total volume is $\frac{4}{3}\pi d^3$.

D. Determine The Lap Length

There is a lap of unknown integer length $L \leqslant 10^9$. We can make queries: run k more meters around the lap, get the total number of completed laps so far. Find L in at most 100 queries.

D. Determine The Lap Length

For an arbitary x, how can we check if $L \leqslant x$? Let D be the total distance we ran so far, and $kx > D$ be the closest multiple of x. Query $kx - D$, and check that the number of laps is $\geqslant k$.

D. Determine The Lap Length

For an arbitary x, how can we check if $L \leqslant x$? Let D be the total distance we ran so far, and $kx > D$ be the closest multiple of x. Query $kx - D$, and check that the number of laps is $\geqslant k$.

Now, binary search, keeping track of the total travelled distance. $\log_2 10^9 \sim 30$ queries.

E. Empires

There is a graph, with vertices divided between three empires. For each empire, build the smallest number of bases in its vertices, so that for each other vertex a base is reachable when vertices of a single other empire become impassable.

E. Empires

Consider empire 1, and block all cities of empire 2. Number the resulting connected components from 1 to X, and for each city i of empire 1 let x_i denote the index of its connected component.

E. Empires

Consider empire 1, and block all cities of empire 2. Number the resulting connected components from 1 to X, and for each city i of empire 1 let x_i denote the index of its connected component.

By blocking empire 3's cities, we produce numbers y_i between 1 and Y in the same way.

E. Empires

Consider empire 1, and block all cities of empire 2. Number the resulting connected components from 1 to X, and for each city i of empire 1 let x_i denote the index of its connected component.

By blocking empire 3's cities, we produce numbers y_i between 1 and Y in the same way.

The task now becomes: choose the smallest number of vertices, such that for each $x = 1, \ldots, X$ at least one vertex with $x_i = x$ is chosen (same for y_i).

E. Empires

Construct a bipartite graph with X and Y vertices in respective halves. For each vertex i, connect vertices x_i and y_i. Note that the graph has $O(n)$ vertices and edges.

E. Empires

Construct a bipartite graph with X and Y vertices in respective halves. For each vertex i, connect vertices x_i and y_i. Note that the graph has $O(n)$ vertices and edges.

We are looking the minimum *edge cover* in this graph. It can be found by taking a maximum matching, and covering remaining vertices with a separate edge each.

E. Empires

Construct a bipartite graph with X and Y vertices in respective halves. For each vertex i, connect vertices x_i and y_i. Note that the graph has $O(n)$ vertices and edges.

We are looking the minimum *edge cover* in this graph. It can be found by taking a maximum matching, and covering remaining vertices with a separate edge each.

Find maximum matching with Kuhn's algorithm. Repeat for empires 2 and 3 similarly. Complexity is $O(m + n^2)$.

F. Finish Time Expectation

For a given convex polygon, find the expected Manhattan distance between uniformly chosen points inside the polygon.

F. Finish Time Expectation

By linearity of expectation, find expected difference between x-coordinates and y-coordinates independently, and sum them up.

F. Finish Time Expectation

By linearity of expectation, find expected difference between x-coordinates and y-coordinates independently, and sum them up.

Consider an infinitesimal segment $[x, x + dx]$. It contributes to the x-distance when one point is to the left of x, and the other is to the right.

F. Finish Time Expectation

By linearity of expectation, find expected difference between x-coordinates and y-coordinates independently, and sum them up.

Consider an infinitesimal segment $[x, x + dx]$. It contributes to the x-distance when one point is to the left of x, and the other is to the right.

Let S be the polygon area, and $L(x)$ be the area to the left of coordinate x. The answer is then equal to

$$\int_{x_{min}}^{x_{max}} \frac{2L(x)(S - L(x))}{S^2} dx.$$

F. Finish Time Expectation

Observe that $L(x)$ is a piecewise linear function between adjacent x-coordinates, thus the integrand is piecewise quadratic. The integral for each piece can then be found analytically.

F. Finish Time Expectation

Observe that $L(x)$ is a piecewise linear function between adjacent x-coordinates, thus the integrand is piecewise quadratic. The integral for each piece can then be found analytically.

Swap x's with y's and repeat to find the expected y-distance.

F. Finish Time Expectation

Observe that $L(x)$ is a piecewise linear function between adjacent x-coordinates, thus the integrand is piecewise quadratic. The integral for each piece can then be found analytically.

Swap x's with y's and repeat to find the expected y-distance.

Big decimals are highly recommended.

G. Generate Optimal Tree

Given are n bit strings of equal length. Build a decision tree of minimum height that can distinguish the given strings by single character lookups.

G. Generate Optimal Tree

Subset DP. For $S \subseteq \{1, \ldots, n\}$, let dp_S be the smallest possible height of a tree distinguishing strings from S.

G. Generate Optimal Tree

Subset DP. For $S \subseteq \{1, \ldots, n\}$, let dp_S be the smallest possible height of a tree distinguishing strings from S.

Let $S_{j,0}, S_{j,1}$ be the partition of S based on the character j. Then we have $dp_S = 1 + \min_j \max(dp_{S_{j,0}}, dp_{S_{j,1}})$ (unless $|S| = 1$, when $dp_S = 0$).

G. Generate Optimal Tree

Subset DP. For $S \subseteq \{1, \ldots, n\}$, let dp_S be the smallest possible height of a tree distinguishing strings from S.

Let $S_{j,0}, S_{j,1}$ be the partition of S based on the character j. Then we have $dp_S = 1 + \min_j \max(dp_{S_{j,0}}, dp_{S_{j,1}})$ (unless $|S| = 1$, when $dp_S = 0$).

Note that if, say, $S_{j,0} = S$, then looking at character j is useless, and such transitions should be skipped.

G. Generate Optimal Tree

Subset DP. For $S \subseteq \{1, \ldots, n\}$, let dp_S be the smallest possible height of a tree distinguishing strings from S.

Let $S_{j,0}, S_{j,1}$ be the partition of S based on the character j. Then we have $dp_S = 1 + \min_j \max(dp_{S_{j,0}}, dp_{S_{j,1}})$ (unless $|S| = 1$, when $dp_S = 0$).

Note that if, say, $S_{j,0} = S$, then looking at character j is useless, and such transitions should be skipped.

The tree for S can be reconstructed by taking $\mathrm{argmin}\ j$ in the recurrence formula, and reconstructing answers for $S_{j,0}, S_{j,1}$.

G. Generate Optimal Tree

Subset DP. For $S \subseteq \{1, \ldots, n\}$, let dp_S be the smallest possible height of a tree distinguishing strings from S.

Let $S_{j,0}, S_{j,1}$ be the partition of S based on the character j. Then we have $dp_S = 1 + \min_j \max(dp_{S_{j,0}}, dp_{S_{j,1}})$ (unless $|S| = 1$, when $dp_S = 0$).

Note that if, say, $S_{j,0} = S$, then looking at character j is useless, and such transitions should be skipped.

The tree for S can be reconstructed by taking $\mathrm{argmin}\ j$ in the recurrence formula, and reconstructing answers for $S_{j,0}, S_{j,1}$.

Complexity $O(2^n n|s|)$, or $O(2^n|s|)$ with bitsets.

H. Heavy Rain

There are n cacti in a row, i-th having height h_i. Process queries: if rain falls on a segment $[L, R]$, how much water will be collected?

H. Heavy Rain

For a query $[L, R]$, water will be kept at height h_i to the left/right of cactus i if the rightmost/leftmost closest cactus j higher or equal than h_i is outside of the segment.

H. Heavy Rain

For a query $[L, R]$, water will be kept at height h_i to the left/right of cactus i if the rightmost/leftmost closest cactus j higher or equal than h_i is outside of the segment.

The water profile is *bitonic*: left part of it is non-decreasing, and right part is non-increasing. For each query, let's find both parts independently.

H. Heavy Rain

To find the left non-decreasing part, use *monotonic stack.* Consider cacti $n, \ldots, 1$, and for the current position L maintain a stack $n = j_1 > \ldots > j_k = L$ of indices of cacti that are higher than any cactus to their left we've seen so far. To introduce cactus $L - 1$, pop several elements from the stack while $h_{j_k} \leqslant h_{L-1}$, and push $L - 1$.

H. Heavy Rain

To find the left non-decreasing part, use *monotonic stack.* Consider cacti $n, \ldots, 1$, and for the current position L maintain a stack $n = j_1 > \ldots > j_k = L$ of indices of cacti that are higher than any cactus to their left we've seen so far. To introduce cactus $L-1$, pop several elements from the stack while $h_{j_k} \leqslant h_{L-1}$, and push $L-1$.

Now, for any query $[L, R]$, cacti in the left part of the profile are a suffix of the monotonic stack. Process queries offline by decreasing of L, and use binary search to find the relevant suffix. If we additionally store prefix sums of $h_{j_s} \times (j_s - j_{s+1})$, we can then compute the area of the left part of the profile. Subtract the range sum of h_i to find out the amount of water.

H. Heavy Rain

Repeat the algorithm in the other direction to find the right part of the profile. If there are several highest cacti between L and R, additionally add water kept between them.

H. Heavy Rain

Repeat the algorithm in the other direction to find the right part of the profile. If there are several highest cacti between L and R, additionally add water kept between them.

Complexity of this solution is $O(n + q \log n)$.

I. Improved Werewolf

Given an array, process queries:

- add x to elements in a range with indices in arithmetic progression spaced 2 or 3;
- find RMQ in a range $[l, r]$;
- erase l-th element;
- insert a previously erased element to its original relative position, and set it to 0.

I. Improved Werewolf

Let's split the array into 6 subarrays, based on their indices modulo 6. The add operation then affects some of the remainders modulo 6, and is a "range add" operation in each of the subarrays.

I. Improved Werewolf

Let's split the array into 6 subarrays, based on their indices modulo 6. The add operation then affects some of the remainders modulo 6, and is a "range add" operation in each of the subarrays.

RMQ can be transformed into 6 RMQ for subarrays.

I. Improved Werewolf

Let's split the array into 6 subarrays, based on their indices modulo 6. The add operation then affects some of the remainders modulo 6, and is a "range add" operation in each of the subarrays.

RMQ can be transformed into 6 RMQ for subarrays.

To erase an element, split the subarrays at its position, rearrange and attach the suffixes accordingly. Insert an element in a similar way.

I. Improved Werewolf

Let's split the array into 6 subarrays, based on their indices modulo 6. The add operation then affects some of the remainders modulo 6, and is a "range add" operation in each of the subarrays.

RMQ can be transformed into 6 RMQ for subarrays.

To erase an element, split the subarrays at its position, rearrange and attach the suffixes accordingly. Insert an element in a similar way.

All this can be done if each of the subarrays is stored in, say, a treap, for $\sim 6 \log n$ operations per query.

J. Juggle Sort

Process given in the statement is equivalent to the following:

Given an array, perform operations sequentially. If the leftmost element is (one of the) largest in the array, erase it. Otherwise, pay 1 coin and move it to the right end. Proceed until the array is empty.

How many coins will be paid?

J. Juggle Sort

Suppose that the array is initially empty. Consider all elements by decreasing, in groups of equal numbers. Insert numbers in each group to their relative positions, and see how the score updates. Maintain a position i of the element that will be erased last, as well as the number c of coins we pay for this element.

J. Juggle Sort

Suppose that the array is initially empty. Consider all elements by decreasing, in groups of equal numbers. Insert numbers in each group to their relative positions, and see how the score updates. Maintain a position i of the element that will be erased last, as well as the number c of coins we pay for this element.

If we insert a new group, its elements will be erased cyclically starting from the leftmost element to the right of i (if any). Elements to right of i infer a cost of c each, while elements to the left i infer cost $c + 1$ each. Thus, we can update the costs, as well as values of i and c.

J. Juggle Sort

Suppose that the array is initially empty. Consider all elements by decreasing, in groups of equal numbers. Insert numbers in each group to their relative positions, and see how the score updates. Maintain a position i of the element that will be erased last, as well as the number c of coins we pay for this element.

If we insert a new group, its elements will be erased cyclically starting from the leftmost element to the right of i (if any). Elements to right of i infer a cost of c each, while elements to the left i infer cost $c + 1$ each. Thus, we can update the costs, as well as values of i and c.

This is easily implemented in $O(n \log n)$ time.

K. King And Toll Roads

There is a graph with n vertices. Vertices i and $i+1$ are adjacent for each $i = 1, \ldots, n-1$ (*trivial* edges), and m extra (*non-trivial*) edges are present.

We can make two edges have cost 1 to travel through. What is the largest sum of pairwise smallest travel costs we can achieve?

K. King And Toll Roads

If the answer is not zero, then removing both toll edges should make the graph disconnected.

K. King And Toll Roads

If the answer is not zero, then removing both toll edges should make the graph disconnected.

Since all vertices are pairwise reachable by trivial edges, at least one of the toll edges should be trivial.

K. King And Toll Roads

If the answer is not zero, then removing both toll edges should make the graph disconnected.

Since all vertices are pairwise reachable by trivial edges, at least one of the toll edges should be trivial.

Consider a few cases:

- Both toll edges are bridges (and thus trivial).

K. King And Toll Roads

If the answer is not zero, then removing both toll edges should make the graph disconnected.

Since all vertices are pairwise reachable by trivial edges, at least one of the toll edges should be trivial.

Consider a few cases:

- Both toll edges are bridges (and thus trivial).
  We can check for each trivial edge $i \leftrightarrow i+1$ if its a bridge with Tarjan's algorithm, or simply by checking that no non-trivial edge xy satisfies $x \leqslant i < i+1 \leqslant y$.
  Suppose we cut away A leftmost vertices and B rightmost vertices. The answer is then $A(n-A)+B(n-B)$.
  The answer is maximized when both toll bridges are closest to the middle $n/2$.

K. King And Toll Roads

- Neither of the toll edges is a bridge, but removing both of them makes the graph disconnected (a *2-bridge*).

K. King And Toll Roads

- Neither of the toll edges is a bridge, but removing both of them makes the graph disconnected (a *2-bridge*).
  Relation “two edges are a 2-bridge” is an equivalence relation.
  Equivalence classes can be found with advanced DFS, or with randomized cycle space approach.

K. King And Toll Roads

- Neither of the toll edges is a bridge, but removing both of them makes the graph disconnected (a *2-bridge*).
  Relation “two edges are a 2-bridge” is an equivalence relation.
  Equivalence classes can be found with advanced DFS, or with randomized cycle space approach.
  In this graph, each equivalence class consists of several trivial edge, and at most one non-trivial edge (when present, it connects leftmost and rightmost parts of the graph with respect to trivial edges).

K. King And Toll Roads

- Neither of the toll edges is a bridge, but removing both of them makes the graph disconnected (a *2-bridge*).
  Relation "two edges are a 2-bridge" is an equivalence relation.
  Equivalence classes can be found with advanced DFS, or with randomized cycle space approach.
  In this graph, each equivalence class consists of several trivial edge, and at most one non-trivial edge (when present, it connects leftmost and rightmost parts of the graph with respect to trivial edges).
  Choosing two best edges in each class reduces to finding a segment in the sequence of part sizes that is closest to $n/2$, and can be done with two pointers.

K. King And Toll Roads

- Neither of the toll edges is a bridge, but removing both of them makes the graph disconnected (a *2-bridge*).
  Relation "two edges are a 2-bridge" is an equivalence relation.
  Equivalence classes can be found with advanced DFS, or with randomized cycle space approach.
  In this graph, each equivalence class consists of several trivial edge, and at most one non-trivial edge (when present, it connects leftmost and rightmost parts of the graph with respect to trivial edges).
  Choosing two best edges in each class reduces to finding a segment in the sequence of part sizes that is closest to $n/2$, and can be done with two pointers.

This results in $O(n \log n)$ solution.

L. LTE Broadcasting Stations

Basically:
There are n points in the real line. We can add directed edge from point x_i to point x_j, paying $f(|x_i - x_j|)$, where $f(D) = D\lfloor\sqrt{D}\rfloor$.
For each $h = 1, \ldots, n-1$, find the smallest cost of constructing a rooted tree with height at most h.

L. LTE Broadcasting Stations

Let $d(x_i)$ be the distance from x_i to the root in the tree.
Let $p(x_i) = x_j$ — the coordinate of the parent of the point x_i (undefined for the root).

L. LTE Broadcasting Stations

Let $d(x_i)$ be the distance from x_i to the root in the tree.
Let $p(x_i) = x_j$ — the coordinate of the parent of the point x_i (undefined for the root).

Claim

In an optimal answer the subtree of each vertex forms a contiguous segment of points.

L. LTE Broadcasting Stations

Let $d(x_i)$ be the distance from x_i to the root in the tree.
Let $p(x_i) = x_j$ — the coordinate of the parent of the point x_i (undefined for the root).

Claim

In an optimal answer the subtree of each vertex forms a contiguous segment of points.

Proof sketch: assuming the contrary, there are points $x_1 < x_2 < x_3$ such that x_1, x_3 are in the subtree of a point x (maybe one of x_1, x_3) , but x_2 is not.

L. LTE Broadcasting Stations

Let $d(x_i)$ be the distance from x_i to the root in the tree.
Let $p(x_i) = x_j$ — the coordinate of the parent of the point x_i (undefined for the root).

Claim

In an optimal answer the subtree of each vertex forms a contiguous segment of points.

Proof sketch: assuming the contrary, there are points $x_1 < x_2 < x_3$ such that x_1, x_3 are in the subtree of a point x (maybe one of x_1, x_3) , but x_2 is not.
WLOG assume $x > x_2$.
If $d(x_2) \leqslant d(x)$, we can improve by making x_2 an ancestor of x_1 by changing its, or one of its ancestors' parent to x.

L. LTE Broadcasting Stations

Let $d(x_i)$ be the distance from x_i to the root in the tree.
Let $p(x_i) = x_j$ — the coordinate of the parent of the point x_i (undefined for the root).

Claim

In an optimal answer the subtree of each vertex forms a contiguous segment of points.

Proof sketch: assuming the contrary, there are points $x_1 < x_2 < x_3$ such that x_1, x_3 are in the subtree of a point x (maybe one of x_1, x_3) , but x_2 is not.
WLOG assume $x > x_2$.
If $d(x_2) \leqslant d(x)$, we can improve by making x_2 an ancestor of x_1 by changing its, or one of its ancestors' parent to x.
If $d(x_2) > d(x)$, we can improve by making x an ancestor of x_2.

L. LTE Broadcasting Stations

We can now find the answer with subsegment DP.

L. LTE Broadcasting Stations

We can now find the answer with subsegment DP.
Let $dp[l, r, h, p]$ be the smallest cost to construct a rooted tree of height at most h, and connect the root of this tree to:

- nothing, if $p = 0$;
- $l - 1$, if $p = 1$;
- $r + 1$, if $p = 2$.

L. LTE Broadcasting Stations

We can now find the answer with subsegment DP.
Let $dp[l, r, h, p]$ be the smallest cost to construct a rooted tree of height at most h, and connect the root of this tree to:

- nothing, if $p = 0$;
- $l - 1$, if $p = 1$;
- $r + 1$, if $p = 2$.

Then, say,

$$dp[l, r, h, 0] = \min_{i=l}^{r} dp[l, i-1, h-1, 2] + dp[i+1, r, h-1, 1],$$

and $p = 1, 2$ differ by an extra summand $f(|x_i - x_{l-1}|)$ or $f(|x_i - x_{r+1}|)$.

L. LTE Broadcasting Stations

We can now find the answer with subsegment DP.
Let $dp[l, r, h, p]$ be the smallest cost to construct a rooted tree of height at most h, and connect the root of this tree to:

- nothing, if $p = 0$;
- $l - 1$, if $p = 1$;
- $r + 1$, if $p = 2$.

Then, say,

$$dp[l, r, h, 0] = \min_{i=l}^{r} dp[l, i-1, h-1, 2] + dp[i+1, r, h-1, 1],$$

and $p = 1, 2$ differ by an extra summand $f(|x_i - x_{l-1}|)$ or $f(|x_i - x_{r+1}|)$.

The answer for height h is $dp[1, n, h, 0]$. This results in $O(n^4)$ complexity.