A

A. Dwarf Tower

There are n different items in the store. One can buy i-th, item at a price c_i, $i = 1, 2, \ldots, n$, or craft item a_j from items x_j and y_j, $j = 1,\ 2,\ \ldots,\ m$. We are to get 1-th item. How much money should we spend for it?

A. Problem A

Let's describe an algorithm, which is quite similar to Dijkstra's algorithm of finding shortest paths.

Declare an array $d = d[1..n]$; let $d[i]$ be minimal current cost of getting (buying or/and crafting) i-th item.

Also, let $finalized = finalized[1..n]$ be boolean array, such that $finalized[i]$ is $true$ iff we are sure that $d[i]$ is optimal price for crafting i.

At the beginning, for each $i \in [1..n]$ $d[i] = c_i$ and $finalized[i] = false$.

A. Dwarf Tower

Our algorithm consists of n steps. On each step:

- let v_k be such item that $d[v_k]$ is minimal among all items with $finalized[v_k] = false$.
- Then, assign $finalized[v]$ to be $true$.
- After that, for all ways of crafting $j \in [1..n]$ such that $x_j = v_k$ or $y_j = v_k$ (and analogically for y_j), try to update $d[a_j]$ by $d[x_j] + d[y_j]$.

This algorithm works in $O(n^2 + m)$ being implemented straightforwardly (TL version) or in $O(m\ log\ n)$ (OK version) if store pairs $\{d[i], i\}$ in std::set or priority queue.

But how to prove that this algorithm works fine?

A. Dwarf Tower

As for Dijkstra's algorithm, proof is based on the following

Lemma A1: suppose that we are now on k-th step of the algorithm. Let S be set of items i such that $d[i] = true$ (before k-th step), and T is a set of all other items. So, v_k will be minimal by $d[v]$ among all $v \in T$. Then:

1. for all items v, $d[v]$ is a cost of such way to create v-th item;
2. for all items $v \in S$, $d[v]$ is optimal;
3. for all items $v \in T$, $d[v]$ does not exceed minimal possible cost of way to craft v using only items of the set S (one of such way is just to buy item v directly).
4. $d[v_k]$ is optimal.

A. Dwarf Tower

To prove Lemma, we'll use method of induction by k.

Base: $k = 0$ - obvious.

Step:

- if we prove last assertion of Lemma, then all others will be just a consequence of this assertion and structure of our algorithm.
- So, consider some way to build some way of crafting v_k. It can be interpreted as a sequence $i_1,\ i_2,\ \ldots,\ l_z = v_k$ of items each of them was bought directly or crafted from two of previous items.
- Suppose that i_l was first item from T in this sequence. Then, i_l was crafted using only elements of S; then, assumption of induction give us the fact that to craft i_l, we spent at least $d[i_l] \geqslant d[v_k]$; then, the cost of the whole algorithm is not less then $d[v_k]$, QED.

B

B. Rays Distance

In this problem, we are to find the distance between ray (p_1, p_2) and ray (p_3, p_4).

B. Rays Distance

Let d_1 be a distance between point p_1 and ray (p_3, p_4), and d_2 be a distance between point p_3 and line (p_1, p_2).

One of the simplest (ideologically) ways to solve the problem is based on the fact that if rays are parallel or don't have common points, then the answer is $\min d_1, d_2$.

To prove this fact, we can consider two points on the rays such that distance between them is the distance between rays. If rays are not parallel, and both points are not origins of their rays, then these points can be moving along the rays in some direction in such a way that direction of line containing these points will be the same during the moving, and the distance will not increasing.

If rays are not parallel, then at some moment of time, at least one of the points will achieve the origin of its ray or points will become equal; it finishes our sketch of proof (case of parallel rays are left to the reader).

B. Rays Distance

So, the algorithm is simple:

- find d_1 and d_2;
- if rays are parallel then return $\min(d_1,\ d_2)$.

- Otherwise, let p be a point of intersection of **lines** (p_1p_2) and (p_3p_4).
- If p lies on both rays, then the answer is 0;
- Otherwise, the answer is $\min(d_1,\ d_2)$.

C

C. Interesting Equation

Given T pairs (A, B); for each pairs, we are to find minimal positive integer N such that N/A is an A-th degree of some integer X and N/B is B-th degree of some integer Y.

C. Interesting Equation

Decompose A and B into prime factors. Suppose that $A = p_1^{a_1}p_2^{a_2}\dots p_k^{a_k}$, $B = q_1^{b_1}q_2^{b_2}\dots q_l^{b_l}$.

Then, the answer will be $ANS = p_1^{c_1}\dots p_k^{c_k}q_1^{d_1}\dots q_l^{d_l}$; here each $c_i(d_j)$ is minimal integer number which is not less then $a_i(b_j)$ and is divided by $B(A)$. Such c_i(case of d_j is left to the reader) can be calculated using Diophantine equation $a_i + c_i * A + z * B = 0$ of two unknown variables c_i and z. Having such c_i-s and d_j-s, all degrees $p_i^{c_i}$ and $q_j^{d_j}$ can be calculated in $O(log(AB))$ using binary exponentation.

Each Diophantine equation can be solved in $O(log(A+B))$ time; so, the complexity of our solution does not exceed $O(log(A+B)log(AB)) = O(log^2(AB))$.

D

D. Race

Given n racers; i-th of them starts from position x_i and goes along straight line with constant speed v_i. We are to find a moment of time when distance between first and last racers are minimal possible.

D. Race

Consider races i and j for some i and j. In system of coordinates connected with i-th racer, j moves with constant speed $v_j - v_i$; so, function $d_{ij}(t)$ equal to distance between i-th and j-th racers at time t is convex down.

Let $d(t)$ is a distance between first and last racers at the moment of time t. Obviously, $d(t) = \max(d_{ij}(t) | 1 \leqslant i < j \leqslant n)$. It means that as maximum of convex down functions, $d(t)$ is convex down; then, minimum of $d(t)$, which is what we are looking for, can be found using ternary search.

For any fixed t, $d(t)$ can be found in $O(n)$ time; then, the complexity of the whole solution is $O(n\ log\ \epsilon^{-1})$.

E. Spies

E. Spies

There are n spies; each of them has a part of secret data nobody else knows. For any two spies i and j, $1 \leqslant i < j \leqslant n$, it's possible to organize a meeting of exchange of all information they have; it costs c_{ij}. After sequence of meetings, we need to choose a group of spies such that these spies knows in total all n parts of information; including of i-th spy in the group costs a_i. What minimal amount of money should we spent?

E. Spies

Build a weighted undirectional graph G in the following way:

- G contains $n+1$ vertices - $1,\ 2,\ \ldots,\ n$ corresponding to spies, and vertex 0 - an additional one.
- For each spy i, vertex 0 is connected with i by an edge of weight a_i.
- Each pair of spies (i, j) is connected by an edge of weight c_{ij}.

Obviously, each sequence of meetings and following choosing of group corresponds to some subgraph G' of G; G' contains all vertices, edges of meetings and edges $(0, p)$ for each chosen spy p; total weight of G' is not less than budget of operation.

E. Spies

On the other hand, minimal spanning tree G'' corresponds to sequence of meetings "from leafs to root 0" and following choosing of group as subset of neighbours of 0; moreover, total weight of G'' is equal to budget of the corresponding operation!

So, the answer is weight of minimal spanning tree of G which can be easily found in $O(n^2)$ time using Prim's algorithm.

F

F. Bacon's filter

In this problem, we are to find a number of different substrings of a string $S = S[1..n]$ of length $n \leqslant 5000$.

F. Bacon's filter

Sort all suffixes $S[i..n]$, $i = 1,\ 2,\ \ldots,\ n$ in $O(n^2)$ lexicographically using bucket sort; let p_1, p_2, ..., p_n is the result sequence, where p_i is number of start of i-th in sorted order suffix.

Then, in $O(n^2)$ we can find $lcp[i]$, $1 \leqslant i < n$, where $lcp[i]$ is length of minimal common prefix of $S[p_i]$ and $S[p_{i+1}]$. After that, it can be easily proved that the answer is $|S[p_0]| + \sum_{i=1}^{n-1}(|S[p_{i+1}]| - lcp[i]) = n - p_0 + 1 + \sum_{i=1}^{n-1}(n - p_{i+1} + 1 - lcp[i]) = n(n+1)/2 - \sum_{i=1}^{n-1} lcp[i]$.

G

G. Hentium Scheduling

On Hentium processor, there are two cores, and one needs to distribute n processes among the cores. It's known that i-th process works a_i time on first core and b_i time on second core. About m pairs (x_j, y_j) of pairs it is known that if processes x_i and y_i are on different cores then connection between them costs c_i time. How to distribute processes with minimal total time of work?

G. Hentium Scheduling

Build a weighted unoriented graph G on n processes and m edges; i-th of them connects process x_i with process y_i and weights c_i. Then:

- add to G vertices s and t.
- After that connect s with each vertex i by edge of weight b_i, $i = 1, 2, ..., n$;
- Then, do the same with t and a_i, $i = 1, 2, ..., n$.

G. Hentium Scheduling

Let V be set of vertices of G. Consider some cut of graph: $V = S \cup T, S \cap T = \emptyset, s \in S, t \in T$. This cut corresponds with some distribution of processes among cores, and total time of work for this distribution is equal to the weight of the cut (by definition, weight of cut is a total weight of all edges connecting vertices from different parts of cut).

Then, to solve a problem, we are to find minimal cut in graph G. This classical problem can be solved by Edmonds-Karp algorithm in $O(nm^2)$.

H

H. Memory Manager

In this problem, one works with memory which consists of N consecutive cells of memory. There are two types of operations to perform:

1. Allocate a segment of given length k in free segment of maximal possible length (in case of several such segments, the leftmost is chosen), starting from the leftmost possible position. In this case, number of left edge of allocated segment need to be printed;
2. Deallocate one of segments allocated before.

H. Memory Manager

Let's store free (maximal by inclusion) segments of memory in two(!!!) std::set's. These sets will contains the same sets of segments, but by different comparators:

1. by leftmost coordinate;
2. by length (and secondly, by leftmost coordinate).

Then, to perform an allocation, one can find optimal free segment as maximum in second set. This segment should be erased from both sets and replaced by right part of segment which remained free (in both sets, too).

H. Memory Manager

To perform deallocation, we should add segment into sets, but before it, using std::set.lower_bound, one can find is it possible to union deallocating segment with some of its neighbours, left and/or right. If so, such neighbours should be erased from both sets, and extended version should be added.

Both allocation and deallocation consist of no more than constant number of queries to std::set, so the complexity of the solution is $O(Mlog\ M)$.

I

I. Heavy chain clusterization

Given n words. Some of them can be united into cluster if they are all have the same prefix of length k or they are all have the same suffix of length k. The problem is to find a way of splitting works into minimal number of clusters.

Firstly, sort all strings lexicographically by their prefix of length k in $O(nk)$ time by bucket sort. Then, list all different k-prefixes; let their amount be A.

After that, do the same for k-suffixes; let B be an amount of different k-suffixes.

The, build a bipartite graph on A prefixes, B suffixes and n edges: for each word, vertices corresponding to k-prefix and k-suffix of the word are connected by an edge. In such graph, any covering set corresponds to a way to clusterise words. So, to solve the problem, we are to find minimal covering number; it can be done in $O((|A|+|B|)n) = O(n^2)$ time using Kuhn's algorithm for search of maximal matching.

J

J. Rectangles

Given N non-degenerate rectangles with sides parallel to axis. Borders of any pair of different rectangles do not intersect. For each rectangle A_i, we are to find a rectangle B_i in which A_I is immediately nested.

J. Rectangles

First of all, compress the coordinates by x-s and by y-s. Then, all x-s and all y-s are from 0 to $2n-1$.

Then, solve the problem by method of vertical scanning line. In details:

- build a segment tree T for elements from 1 to $2n-1$; init all cells by 0.
- for each i-th, $1 \leqslant i \leqslant n$, rectangle with coordinates $(x_1, y_1, x_2, y_2), x_1 < x_2,\ y_1 < y_2$, build two **events** - four of number $\{x,\ y_1, y_2,\ flag,\ number\}$:
  - opening - $\{x_1, y_1, y_2, 0, i\}$;
  - closing - $\{x_1, y_1, y_2, 1, i\}$.

  After that, we have an array evs of $2n$ such events.
- Sort array evs by x-s. Then, go through sorted evs.

J. Rectangles

- Denote current event from evs as ev.
- If ev is opening ($ev.flag = 0$), then:
  - get value of $evs.y_2$-th cell of tree T; save it as answer for $evs.number$ rectangle;
  - assign cells of T from $evs.y_1 + 1$ to $evs.y_2$ to $evs.number$.
- If ev is closing, then just assign cells of T from $evs.y_1 + 1$ to $evs.y_2$ to an answer, found for $evs.number$-th rectangle earlier.

The complexity of the whole solution is $O(n\ log\ n)$.

K

K. Problem K

Given connected unoriented graph on n vertices and m edges, and we are to orient as many edges as possible in such a way that graph remains equal.

K. Problem K

It's obvious that if edge is a bridge then this edge cannot be oriented.

Bridges can be found by Taryan's algorithm in $O(n+m)$; do it.

Then, run DFS on the graph and orient non-bridge-edges along direction of first visiting of the edge.

It can be proven that if graph does not contain bridges then oriented version of the graph will be strongly connected.

It means that if graph has more than one bridge and, as a consequence, more than one component of edge biconnectivity, then nevertheless, our DFS transform each of such component into strongly connected graph.

Total complexity of our solution is $O(n+m)$.

L

L. Internet Shop

In some internet shop, we (with Mihail) want to make n purchases with costs p_1, p_2, ..., p_n thalers, strongly in this order; all p_i-s don't exceed $m = 10^5$. Each purchase can be paid using bonuses or not. If we don't use bonuses to pay for purchase i then we get p_i/C bonuses, $C = 100$; otherwise, we can spend any number of bonuses we have instead of thalers. What minimal amount of money should we spend?

L. Internet Shop

One of key ideas of the solution is that if we decided to spend our bonuses to pay for i-th purchase, then we should spend as maximal number of bonuses as possible.

Let's prove this fact. Suppose that on this step, we spend some non-zero bonuses and could spend at least one bonus more, but we didn't do that. Then, there are two possible variants:

- this one bonus will never be used; then, it's more optimal for us to use the bonus;
- this one bonus will be use later. Then, let's spend this bonus now instead of place we should use this bonus in current version. It doesn't change total amount of money we spend.

These variants mean that it makes no sense to save bonuses during particular purchases if we decided to use them.

L. Internet Shop

Then, use forward dynamic programming. Let $dp[i][b]$, $0 \leqslant i \leqslant n$, $0 \leqslant b \leqslant nm/C$ be minimal amount of money we are to spend for first i purchases in such a way that after paying for i-th purchase, we have exactly b bonuses.

Then, from each state $\{i, b\}$ we can go to state $\{i+1, b+p_{i+1}/100\}$ (paying for $i+1-th$ purchase without using bonuses) or $\{i+1, b-min(b, p_{i+1})\}$ (with using bonuses); both transitions can be processed in $O(1)$. The answer will be $min\{d[n][b] | 0 \leqslant b \leqslant \ nm/C\}$.

L. Internet Shop

The complexity of our solution is $O(n^2 m/C)$.

M

M. Electricity

There two incompatible standards, A and B, for plugs and sockets. We have one main A-type socket, n AB-strips and m BA-strips. i-th of AB-strips has one plug of type A and a_i sockets of type B, $1 \leqslant i \leqslant n$. j-th of BA-strips has one plug of type B and b_j sockets of type A, $1 \leqslant j \leqslant m$. The problem is to build a network with maximum number of sockets of type A.

M. Electricity

Sort arrays a and b by decreasing (in fact, non-increasing) of numbers. Then, let's try to find the answer in a greedy way.

At the beginning, we have only one socket of type A and none of sockets of type B; also, no AB- or BA-strips are used. Also, let's store our current answer in variable ans. At the beginning, $ans = 1$.

At first step, let plug AB-strip with maximal number, i.e. a_1 of B-sockets, into A-socket which we have. After that, try to plug maximal possible number of BA-strips with maximal total number of A-sockets into our plug-in. After that, try to update an answer by current number of available A-steps.

M. Electricity

At k-th of next step, we have x_k available A-sockets in a construction with $k-1$ first (in array a) AB-strips and j_k first (in array b) BA-strips.

If $x_k = 0$, or there are no non-used strips (AB- or BA-) then stop the process. If no, try to plug k-th AB-strip into one of A-socket. Then, greedy plug non-used BA-strips starting from j_k+1 into free B-sockets in our construction. And of course, don't forget to update an answer during this process!

If store and support j_k in explicit way then process can be emulated in $O(n+m)$ time in total; so, total complexity of our solution is $O(n \ log \ n \ + m \log \ m)$.

N

N. Piatra Neamt

Given a tree with n vertices, we are to find vertex v (or a couple of such vertices) such that minimal possible sum of distances between v and all other vertices is minimal possible.

N. Piatra Neamt

Let $sum(u)$ be a sum of distances between u and all possible vertices. We will find $sum(u)$ for **each** of vertices (having that sum, the answer could be found easily).

Let our tree be rooted with, for example, root 0. Then, run DFS which for each vertex v calculates:

- $size(v)$, number of vertices in subtree with root as v;
- $sum2(v)$, sum of distances between v and all vertices of the subtree of v.

After running this DFS, $sum2(0)$ is equal to $sum(v)$!

Of course, we can do the same for each vertex, not only for 0, but in this case, the complexity of the solution will be $O(n^2)$ which is too slow for $n = 100000$. How to find all (except $sum(0)$) $sum(v)$ faster?

N. Piatra Neamt

Run second DFS from root 0. During this DFS, we assume that if DFS entered vertex v, then $sum(v)$ has been calculated already. So, for each child u of v, we are to calculate $sum(u)$ in assumption that we know $sum(v)$, and also $sum2(u)$ and $size(u)$.

Let d be sum of distances from v to all vertices of the tree except u and its descendants. It's easy to see that $d = sum(v) - (sum(u) + size(u))$.

Then, observe that $sum(u) = sum2(u) + d + (n - size(u))$ $(d + (n - size(u))$ is sum of distances between u and vertices which are not descendants of u). So, having $sum(v)$, $sum2$ and $size$, $sum(u)$ can be calculated in $O(1)$. Save $sum(u)$ and run recursive launch of $DFS(u)$.

So, our algorithm consists of two DFSs, and the complexity of the solution is $O(n)$.