Editorial

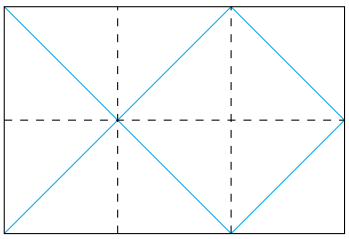
Problem A

First compute the status of the light when you arrive if you go at maximum speed. If the light is green then you have to go at that speed because you want to arrive as soon as possible. If it is red, you can go slower. Compute the first time at which the light becomes green after that and chose your speed so that you travel distance d is exactly that time.

Problem B

If we divide the table into squares of size $d = \gcd(h, w)$ we notice that the ball traverses the diagonal of every square before it reaches the first corner. Thus the total distance is equal to length of each of those diagonals ($\sqrt{2}d$) multiplied by the total numbers of squares ($(h/d) \cdot (w/d)$).

So the answer is: $\frac{dhw\sqrt{2}}{d^2} = \sqrt{2}\frac{hw}{d}$.



Problem C

This problem can easily be to a weighted interval scheduling problem. Each car corresponds to an interval from the time it arrives until the time it leaves the bridge and its cost is the value payed.

We want to select a subset of intervals such that no two intervals properly intersect and the total cost of the chosen intervals is maximum. This is easily solvable with dynamic programming.

$$dp(I) = \text{max value for intervals in set } I$$

Given an interval x let $c(x)$ be the set of intervals conflicting with x. Let $v(x)$ the value value of an interval x. We can write:

$$dp(I) = \max \begin{cases} v(x) + dp(I \setminus c(x)) & \text{take interval } x \\ dp(I \setminus \{x\}) & \text{do not take interval } x \end{cases}$$

Now, this DP formulation is not very good because it contains as many states as there are subsets of I ($2^{|I|}$).

We can do better by sorting the intervals by right end $x_1, x_2, \ldots, x_n$. Then it is easy to see that $c(x_i)$ is a continuous range $x_j, x_j + 1, \ldots, x_i$ for some x_j with $j < i$. This x_j will be the rightmost interval that does not intersect x_i and can be found in $O(\log(n))$ by doing a binary search. Let's write $j(i)$ to be the index of the rightmost interval $x_{j(i)}$ that does not conflict with x_i. Then we can write

$$dp(i) = \text{max value for intervals in set } x_1, x_2, \ldots, x_i$$

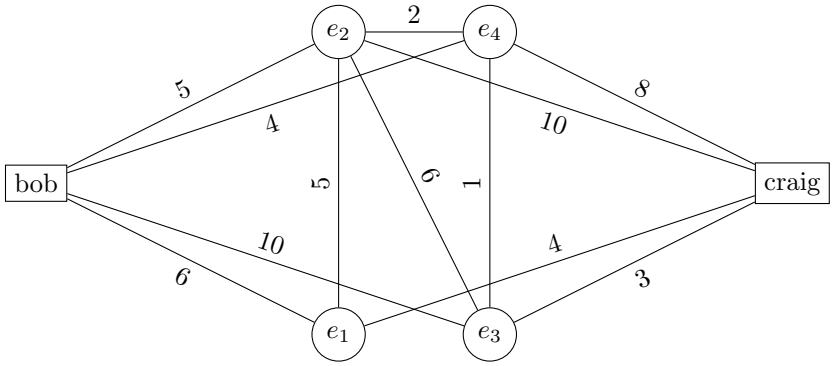
and

$$dp(i) = \max \begin{cases} v(x) + dp(j(i)) & \text{take interval } x_i \\ dp(i-1) & \text{do not take interval } x_i \end{cases}$$

Since each $j(i)$ can be computed in $O(\log(n))$ the total runtime will be $O(n \log(n))$.

Problem D

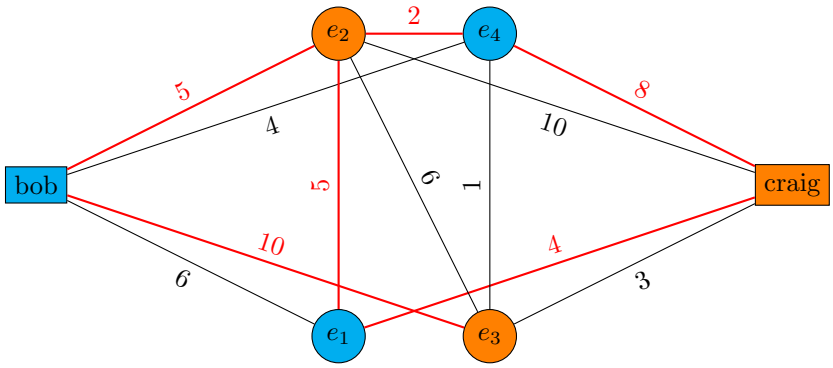
Build a graph with $n + 2$ nodes. One node per exercises and one node representing Bob and another Craig. Link the exercises with an edge with cost equal to the time it take to combine their solutions if they are not solved by the same person. Link Bob to each exercise with and edge of cost equal to the time Bob takes to solve that exercise. Do the same with Craig. The graph in the first sample input will be:



It is not hard to see that there is a one-to-one correspondence between $(bob, craig)$ cuts in this graph and homework assignments. Let's see this on an example. Consider the cut:

$$bob,\ e_4,\ e_1 \ \Big|\ craig,\ e_2,\ e_3$$

Look at the edges crossing the cut (in red):



The cut edges are $(bob, e_2), (bob, e_3), (craig, e_1), (craig, e_4), (e_2, e_4)$ and (e_1, e_2). If we assign e_2, e_3 to *bob* and e_1, e_4 to *craig* the total cost will be exactly the same as the cost of this cut (notice that we reverse the assignment relative to the cut).

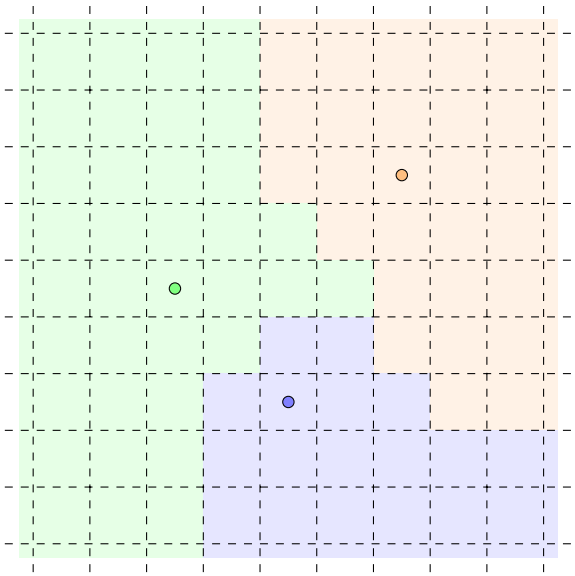
It is not hard to show that this works in general. So the solution is to compute the minimum $(bob, craig)$. The exercises that are on the same side of *bob* in the cut are assigned to *craig* and the other to *bob*.

Problem E

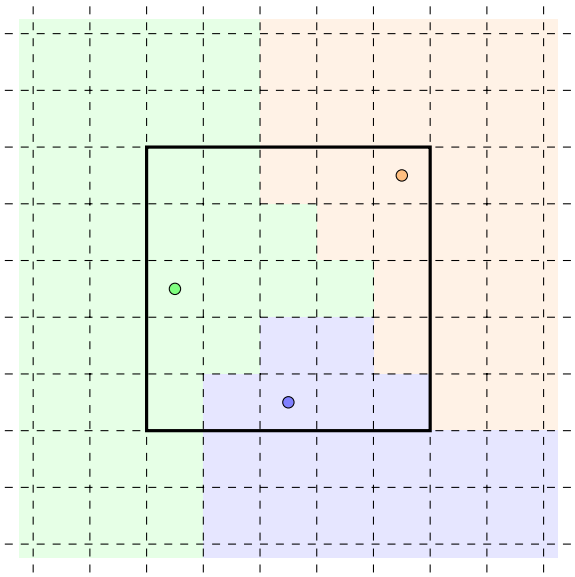
We will pre-compute a data structure that gives for every point, the closest point. This is a sort of Voronoi diagram for the Manhattan distance. Computing

a Voronoi is complicated in general but in this case the problem constraints make it simpler.

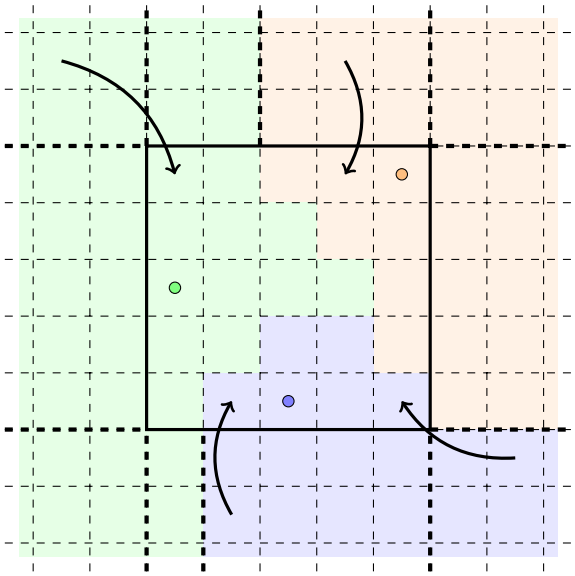
The diagram in this case can be a matrix such that each cell tell which is the closest store from that point. The following figure shows an example where each cell is labeled with the color of the closest store.



Computing such a matrix is easy, a simple BFS from the set of stores does the job. However the coordinates can be very large and the matrix will not fit into memory. If we look at it carefully, the only part where the matrix is messy is between all the stores.



Outside of the bounding box of the stores (from (min_x, min_y) to (max_x, max_y)) each query can be projected to the border of that bounding box.



Because we know that $dx \cdot dy \leq 10^6$ our BFS to compute the Voronoi diagram on the bounding box will run fast enough. Then each query can be answered in $O(1)$ by either reporting the matrix value if it lies inside the bounding bot or projecting it onto the border otherwise.