# LCC Takeup

By Max, Peter, Kenneth, and ChrisT

## LCC ‘20 Contest 3 J1 - Cup Swapping

Start with an empty array of 3 elements, and represent the ball by setting the corresponding element of the array.

For each of the K swaps, swap 2 the elements in the array. Swapping can be done with a temporary variable to assign their values to each other.

Then loop through the 3 elements of the array to find out where the ball is.

Alternatively, use 3 variables (for the first, second, and third cup) instead of an array.

## LCC ‘20 Contest 3 J2 - Tracy and Big Mac(beth)

Store every hamburger in an array or a string. In many languages, this can be done with a method called “split()”.

Add 1 if the first element is a bun and add 1 if the last element is a bun. Notice that the only element of an array of size 1 is the first and last element. Loop through every element that isn’t the first or last and subtract 1 for every bun.

Loop through every element and add 1 for each meat, up until the first cheese. This can be done by breaking out of the loop when seeing a cheese.

Loop through every element and maintain a boolean variable for whether there was sauce.

Loop through every element and add 1 for every pickle and subtract 1 for a feces or rat.

## LCC ‘20 Contest 3 J3 - One-Trick Pony

Notice how the pickrates and winrates are given as ratios.

The pickrate of champion i, $P_i = p[i] / \Sigma p[i]$

The winrate of champion i, $W_i = w[i] / \Sigma w[i]$

Calculate the expected value (average winrate) for all champions: $\Sigma P_i * W_i$

Now calculate the expected value of all champions except champion j:

$E_j = (\Sigma P_i - Pj) / (\Sigma W_i - W_j)$

Ban the champion that maximizes $E_j$.

## LCC ‘20 Contest 3 J4 - Chain Lightning

**Problem Statement:**
You are given an array of elements with a position and a point value. The array is sorted by position. A ”chain” of elements is defined as a continuous sequence where the distance between elements is at most a distance of x and contains at most k elements. Find the maximum point value of a singular chain.

Notice as all of the given point values are positive, for a given chain of elements where one is a subset of another. You only have to consider the maximum chain. For example, taking elements 1, 2 and 3 is always more more ideal than taking just elements 2 and 3.

## LCC ‘20 Contest 3 J4 - Chain Lightning

From our previous observation, we reduce the number of operations we have to complete as we don’t have to check subsets. What do we do in the case where the set of elements in reach is greater than **K**?

Let’s say K is 3 and we have 5 elements in reach. The possible subsets would then be [1, 2, 3], [2, 3, 4] and [3, 4, 5]. We add and remove one element from each subsequent subset. To complete the sum query in O(1) time, sliding window or PSA can be utilized.

The maximum of all sum queries is our answer.

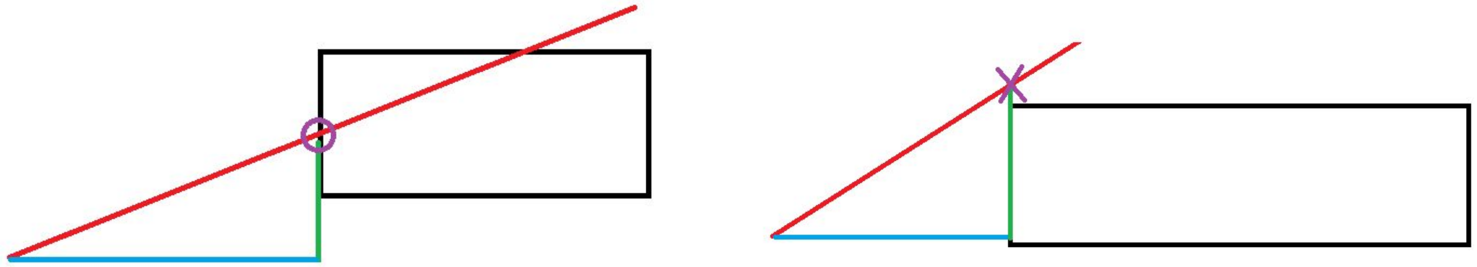
## LCC ‘20 Contest 3 J5 - Raytracing

First, notice that the edges of the rectangle are either horizontal or vertical, and that a rectangle has 4 edges.

We will find the entering and exiting times by calculating the time it takes for the line Andy makes to intersect with each of the edges. Ignore any of the edges that Andy does not cross.

To see when Andy will cross an edge, remember that the edges are either horizontal or vertical. For example, to find out the time to cross a vertical edge, divide the horizontal displacement to the edge by Andy’s horizontal velocity.

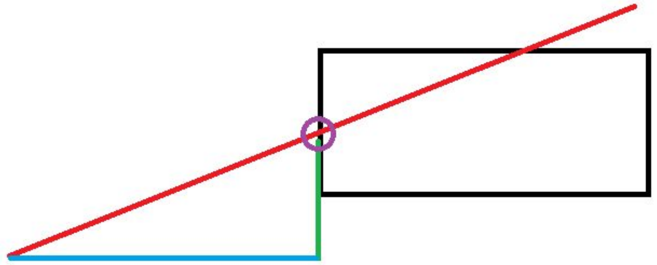
# LCC ‘20 Contest 3 J5 - Raytracing

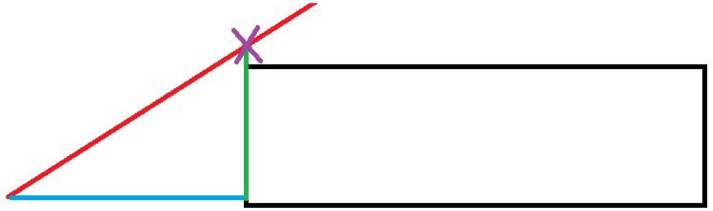


By dividing the horizontal displacement by its velocity we can find the amount of time until a point is on the same x-coord as the start of the box. With the given time we can calculate whether or not it intersects, by increasing our y-coord accordingly. Illustrated above are two possible cases.

Sort the times it takes for Andy to cross each edge (ignoring the edges Andy does not cross) in an array. Note that the length of the array will always be even and not greater than 4.

## LCC ‘20 Contest 3 J5 - Raytracing



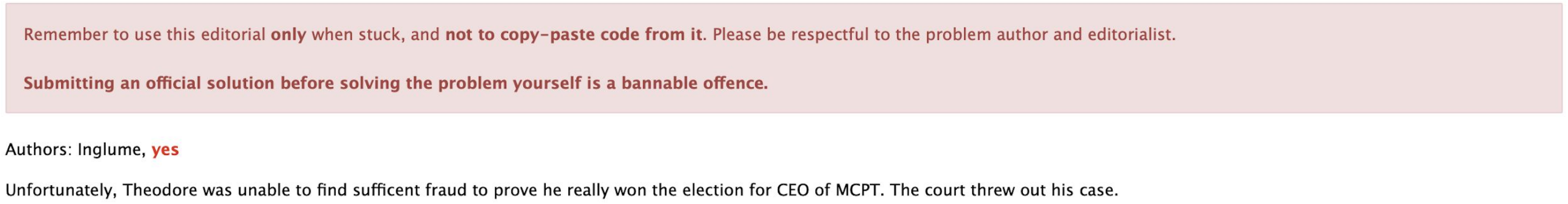


If Andy does not start inside the rectangle or if Andy enters and exits within the same second, print “-1 -1”.

If Andy starts inside the rectangle, then the exiting time is the first nonnegative value. Remember that an integer valued exiting time means that it is still in the rectangle.

If Andy starts outside the rectangle, then the entering time is the first valid line intersection, and the exiting time is the second valid line intersection. Remember that some of the line intersections are not actually on the rectangle.

## LCC '20 Contest 3 S1 - Peculiar Polling



Remember to use this editorial **only** when stuck, and **not to copy-paste code from it**. Please be respectful to the problem author and editorialist.

**Submitting an official solution before solving the problem yourself is a bannable offence.**

Authors: Inglume, **yes**

Unfortunately, Theodore was unable to find sufficent fraud to prove he really won the election for CEO of MCPT. The court threw out his case.

## LCC ‘20 Contest 3 S1 - Peculiar Polling

Consecutive votes for the same candidate count as one vote, so only count the vote if the previous letter is different than the current one.

**ABBBABBBABBBABBBABBBAA** → **ABABABABABABA**

Loop through and increment a counter if s[i] != s[i-1]

## LCC ‘20 Contest 3 S2 - Water Buckets

We are given updates of the form (L,R,c) ⟶ a[i] += c for all L <= i <= R, and we want to determine the first time a[i] >= k for each i. By line sweeping over the indices, we can maintain a list of (c,t) pairs for each index that indicates a[i] += c at time t. Once we have this, we merely want to binary search for the first time such that the total added by that time is >= K. You can do this with a binary-indexed tree or a segment tree.

## LCC ‘20 Contest 3 S3 - Snake

The intended solution to this problem is brute force. Consider all starting points, and maintain the set of visited cells while doing some kind of backtracking. Implementation may differ, but basically anything passes for these constraints.

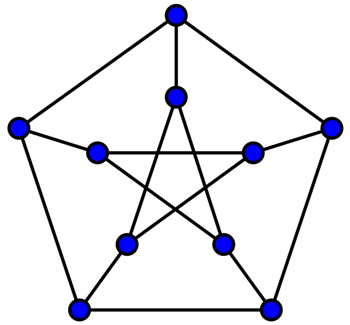
## LCC ‘20 Contest 3 S4 - Minimum Maximum Edge

Consider binary searching over the answer ‘x’. For each edge, we know the number of operations we must use on this edge for its weight to be <= x, ‘c’. Clearly, the optimal edges to take are the ones with minimal c. Specifically, we want to take the minimum spanning tree weighted by c. If the sum of the c values is <= K, then the answer of x is possible, otherwise it is not.

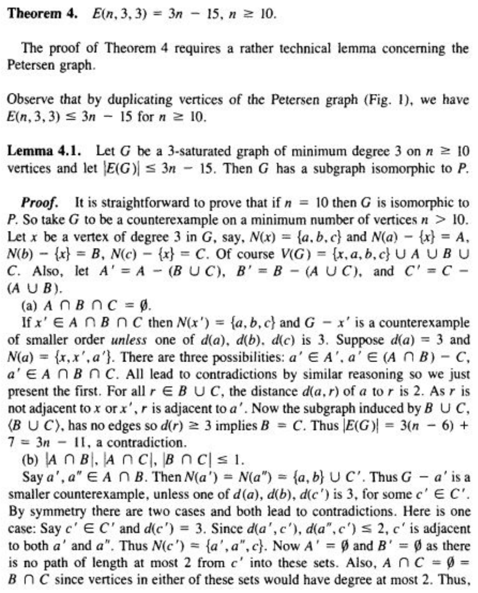
## LCC ‘20 Contest 3 S5 - Larry and 3

For 6 <= n < 10, the intended solution is to solve it by hand. Alternatively, you can write a brute force locally and determine the solutions that way.

For n >= 10, we claim the minimal number of edges is 3n-15. We claim that any graph with n >= 10 that satisfies the properties has a subgraph that is isomorphic to the Petersen graph (pictured below), and that to add k nodes to the Petersen graph such that it still satisfies the properties requires at least 3k edges. Proof is left as an exercise.



# LCC ‘20 Contest 3 S5 - Larry and 3



**Theorem 4.** $E(n,3,3) = 3n - 15$, $n \geq 10$.

The proof of Theorem 4 requires a rather technical lemma concerning the Petersen graph.

Observe that by duplicating vertices of the Petersen graph (Fig. 1), we have $E(n,3,3) \leq 3n - 15$ for $n \geq 10$.

**Lemma 4.1.** Let $G$ be a 3-saturated graph of minimum degree 3 on $n \geq 10$ vertices and let $|E(G)| \leq 3n - 15$. Then $G$ has a subgraph isomorphic to $P$.

***Proof.*** It is straightforward to prove that if $n = 10$ then $G$ is isomorphic to $P$. So take $G$ to be a counterexample on a minimum number of vertices $n > 10$. Let $x$ be a vertex of degree 3 in $G$, say, $N(x) = \{a,b,c\}$ and $N(a) - \{x\} = A$, $N(b) - \{x\} = B$, $N(c) - \{x\} = C$. Of course $V(G) = \{x,a,b,c\} \cup A \cup B \cup C$. Also, let $A' = A - (B \cup C)$, $B' = B - (A \cup C)$, and $C' = C - (A \cup B)$.

(a) $A \cap B \cap C = \emptyset$.

If $x' \in A \cap B \cap C$ then $N(x') = \{a,b,c\}$ and $G - x'$ is a counterexample of smaller order *unless* one of $d(a)$, $d(b)$, $d(c)$ is 3. Suppose $d(a) = 3$ and $N(a) = \{x,x',a'\}$. There are three possibilities: $a' \in A'$, $a' \in (A \cap B) - C$, $a' \in A \cap B \cap C$. All lead to contradictions by similar reasoning so we just present the first. For all $r \in B \cup C$, the distance $d(a,r)$ of $a$ to $r$ is 2. As $r$ is not adjacent to $x$ or $x'$, $r$ is adjacent to $a'$. Now the subgraph induced by $B \cup C$, $\langle B \cup C\rangle$, has no edges so $d(r) \geq 3$ implies $B = C$. Thus $|E(G)| = 3(n-6) + 7 = 3n - 11$, a contradiction.

(b) $|A \cap B|$, $|A \cap C|$, $|B \cap C| \leq 1$.

Say $a', a'' \in A \cap B$. Then $N(a') = N(a'') = \{a,b\} \cup C'$. Thus $G - a'$ is a smaller counterexample, unless one of $d(a)$, $d(b)$, $d(c')$ is 3, for some $c' \in C'$. By symmetry there are two cases and both lead to contradictions. Here is one case: Say $c' \in C'$ and $d(c') = 3$. Since $d(a',c')$, $d(a'',c') \leq 2$, $c'$ is adjacent to both $a'$ and $a''$. Thus $N(c') = \{a', a'', c\}$. Now $A' = \emptyset$ and $B' = \emptyset$ as there is no path of length at most 2 from $c'$ into these sets. Also, $A \cap C = \emptyset = B \cap C$ since vertices in either of these sets would have degree at most 2. Thus,

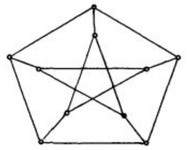
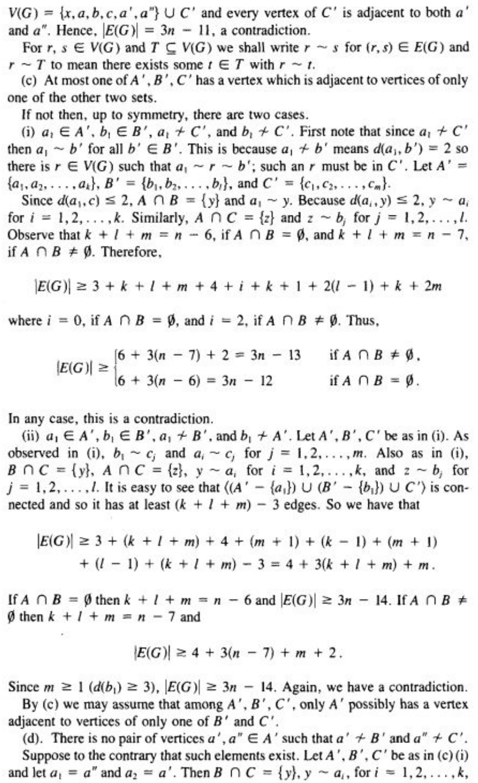


FIGURE 1.



$V(G) = \{x,a,b,c,a',a''\} \cup C'$ and every vertex of $C'$ is adjacent to both $a'$ and $a''$. Hence, $|E(G)| = 3n - 11$, a contradiction.

For $r$, $s \in V(G)$ and $T \subseteq V(G)$ we shall write $r \sim s$ for $(r,s) \in E(G)$ and $r \sim T$ to mean there exists some $t \in T$ with $r \sim t$.

(c) At most one of $A', B', C'$ has a vertex which is adjacent to vertices of only one of the other two sets.

If not then, up to symmetry, there are two cases.

(i) $a_1 \in A'$, $b_1 \in B'$, $a_1 \not\sim C'$, and $b_1 \not\sim C'$. First note that since $a_1 \not\sim C'$ then $a_1 \sim b'$ for all $b' \in B'$. This is because $a_1 \not\sim b'$ means $d(a_1, b') = 2$ so there is $r \in V(G)$ such that $a_1 \sim r \sim b'$; such an $r$ must be in $C'$. Let $A' = \{a_1, a_2, \ldots, a_k\}$, $B' = \{b_1, b_2, \ldots, b_l\}$, and $C' = \{c_1, c_2, \ldots, c_m\}$.

Since $d(a_1, c) \leq 2$, $A \cap B = \{y\}$ and $a_1 \sim y$. Because $d(a_i, y) \leq 2$, $y \sim a_i$ for $i = 1,2,\ldots,k$. Similarly, $A \cap C = \{z\}$ and $z \sim b_j$ for $j = 1,2,\ldots,l$. Observe that $k + l + m = n - 6$, if $A \cap B = \emptyset$, and $k + l + m = n - 7$, if $A \cap B \neq \emptyset$. Therefore,

$$|E(G)| \geq 3 + k + l + m + 4 + i + k + 1 + 2(l-1) + k + 2m$$

where $i = 0$, if $A \cap B = \emptyset$, and $i = 2$, if $A \cap B \neq \emptyset$. Thus,

$$|E(G)| \geq \begin{cases} 6 + 3(n-7) + 2 = 3n - 13 & \text{if } A \cap B \neq \emptyset, \\ 6 + 3(n-6) = 3n - 12 & \text{if } A \cap B = \emptyset. \end{cases}$$

In any case, this is a contradiction.

(ii) $a_1 \in A'$, $b_1 \in B'$, $a_1 \not\sim B'$, and $b_1 \not\sim A'$. Let $A', B', C'$ be as in (i). As observed in (i), $b_1 \sim c_j$ and $a_i \sim c_j$ for $j = 1,2,\ldots,m$. Also as in (i), $B \cap C = \{y\}$, $A \cap C = \{z\}$, $y \sim a_i$ for $i = 1,2,\ldots,k$, and $z \sim b_j$ for $j = 1,2,\ldots,l$. It is easy to see that $\langle (A' - \{a_1\}) \cup (B' - \{b_1\}) \cup C' \rangle$ is connected and so it has at least $(k + l + m) - 3$ edges. So we have that

$$|E(G)| \geq 3 + (k + l + m) + 4 + (m + 1) + (k - 1) + (m + 1) \\ + (l - 1) + (k + l + m) - 3 = 4 + 3(k + l + m) + m\,.$$

If $A \cap B = \emptyset$ then $k + l + m = n - 6$ and $|E(G)| \geq 3n - 14$. If $A \cap B \neq \emptyset$ then $k + l + m = n - 7$ and

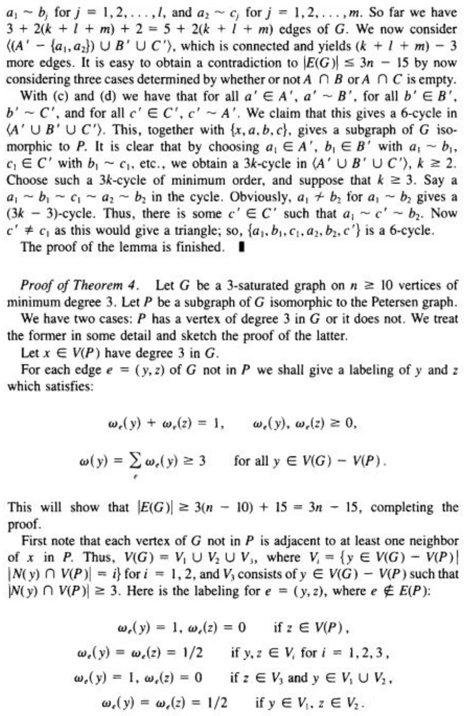
$$|E(G)| \geq 4 + 3(n-7) + m + 2\,.$$

Since $m \geq 1$ ($d(b_1) \geq 3$), $|E(G)| \geq 3n - 14$. Again, we have a contradiction.

By (c) we may assume that among $A', B', C'$, only $A'$ possibly has a vertex adjacent to vertices of only one of $B'$ and $C'$.

(d). There is no pair of vertices $a', a'' \in A'$ such that $a' \not\sim B'$ and $a'' \not\sim C'$.

Suppose to the contrary that such elements exist. Let $A', B', C'$ be as in (c)(i) and let $a_1 = a''$ and $a_2 = a'$. Then $B \cap C = \{y\}$, $y \sim a_i$, for $i = 1,2,\ldots,k$,



$a_1 \sim b_j$ for $j = 1,2,\ldots,l$, and $a_2 \sim c_j$ for $j = 1,2,\ldots,m$. So far we have $3 + 2(k + l + m) + 2 = 5 + 2(k + l + m)$ edges of $G$. We now consider $\langle (A' - \{a_1, a_2\}) \cup B' \cup C' \rangle$, which is connected and yields $(k + l + m) - 3$ more edges. It is easy to obtain a contradiction to $|E(G)| \leq 3n - 15$ by now considering three cases determined by whether or not $A \cap B$ or $A \cap C$ is empty.

With (c) and (d) we have that for all $a' \in A'$, $a' \sim B'$, for all $b' \in B'$, $b' \sim C'$, and for all $c' \in C'$, $c' \sim A'$. We claim that this gives a 6-cycle in $\langle A' \cup B' \cup C' \rangle$. This, together with $\{x,a,b,c\}$, gives a subgraph of $G$ isomorphic to $P$. It is clear that by choosing $a_1 \in A'$, $b_1 \in B'$ with $a_1 \sim b_1$, $c_1 \in C'$ with $b_1 \sim c_1$, etc., we obtain a $3k$-cycle in $\langle A' \cup B' \cup C' \rangle$, $k \geq 2$. Choose such a $3k$-cycle of minimum order, and suppose that $k \geq 3$. Say $a_1 \sim b_1 \sim c_1 \sim a_2 \sim b_2$ in the cycle. Obviously, $a_1 \not\sim b_2$ for $a_1 \sim b_2$ gives a $(3k-3)$-cycle. Thus, there is some $c' \in C'$ such that $a_1 \sim c' \sim b_2$. Now $c' \neq c_1$ as this would give a triangle; so, $\{a_1, b_1, c_1, a_2, b_2, c'\}$ is a 6-cycle.

The proof of the lemma is finished. ∎

*Proof of Theorem 4.* Let $G$ be a 3-saturated graph on $n \geq 10$ vertices of minimum degree 3. Let $P$ be a subgraph of $G$ isomorphic to the Petersen graph.

We have two cases: $P$ has a vertex of degree 3 in $G$ or it does not. We treat the former in some detail and sketch the proof of the latter.

Let $x \in V(P)$ have degree 3 in $G$.

For each edge $e = (y,z)$ of $G$ not in $P$ we shall give a labeling of $y$ and $z$ which satisfies:

$$\omega_e(y) + \omega_e(z) = 1, \qquad \omega_e(y), \omega_e(z) \geq 0,$$

$$\omega(y) = \sum_e \omega_e(y) \geq 3 \qquad \text{for all } y \in V(G) - V(P)\,.$$

This will show that $|E(G)| \geq 3(n-10) + 15 = 3n - 15$, completing the proof.

First note that each vertex of $G$ not in $P$ is adjacent to at least one neighbor of $x$ in $P$. Thus, $V(G) = V_1 \cup V_2 \cup V_3$, where $V_i = \{y \in V(G) - V(P) \mid |N(y) \cap V(P)| = i\}$ for $i = 1, 2$, and $V_3$ consists of $y \in V(G) - V(P)$ such that $|N(y) \cap V(P)| \geq 3$. Here is the labeling for $e = (y,z)$, where $e \notin E(P)$:

$$\begin{aligned} \omega_e(y) = 1,\ \omega_e(z) = 0 &\qquad \text{if } z \in V(P)\,, \\ \omega_e(y) = \omega_e(z) = 1/2 &\qquad \text{if } y, z \in V_i \text{ for } i = 1,2,3\,, \\ \omega_e(y) = 1,\ \omega_e(z) = 0 &\qquad \text{if } z \in V_3 \text{ and } y \in V_1 \cup V_2\,, \\ \omega_e(y) = \omega_e(z) = 1/2 &\qquad \text{if } y \in V_1,\ z \in V_2\,. \end{aligned}$$

# LCC '20 Contest 3 S5 - Larry and 3

It is obvious that for all $z \in V_3$, $\omega(z) \geq 3$. Also, if $y \in V_2$ then with its neighbors in $P$ it is within two of seven of the vertices of $P$, while the remaining three vertices of $P$ form a path in $P$. (See Fig. 2.)

Thus, $y$ is adjacent to at least two other vertices of $V(G) - V(P)$, and $\omega(y) \geq 3$. Consider $u \in V_1$ as in Fig. 3.

In order for $u$ to be within two of all vertices of $P$, *either* $u$ is adjacent to at least two vertices of $V_3$, *or* $u$ is adjacent to at least four vertices of $V_1 \cup V_2$, *or* $u$ is adjacent to one vertex of $V_3$ and at least two of $V_1 \cup V_2$, *or* $u$ is adjacent to exactly three vertices of $V_2$. Only in the last case is $\omega(u) \ngeq 3$. So assume $u$ is adjacent to just $a_1, a_2, a_3 \in V_2$. Since $u$ is adjacent to a neighbor of $x$, with no loss of generality we have the situation in Fig. 3. Also, $N(a_i) \cap N(x) \neq \emptyset$ so $a_i \sim \{r, s\}$ for $i = 1, 2, 3$. This leaves four vertices in $P$, $V(P) - (N(t) \cup \{r, s, t\})$, which each must be adjacent to at least one of $a_1, a_2, a_3$. This is impossible as $a_i \in V_2$ and $a_i \sim \{r, s\}$ for $i = 1, 2, 3$.

This completes the proof of the first case. Suppose that $P$ has no vertex of degree 3 and let $x$ have degree 3 in $G$. Then $x$ is adjacent to 3, 2, 1, or 0 vertices of $P$.

If $x$ is adjacent to 3 vertices of $P$ then $N(x) = N(x')$ for some $x' \in V(P)$. Let $P'$ have vertex set $(V(P) - \{x'\}) \cup \{x\}$ and proceed as in the first case with $P'$ in place of $P$.

If $x$ is adjacent to two vertices of $P$ then, without considering its remaining neighbor in $G$, $x$ is within two of seven vertices of $P$ while the remaining three form a path in $P$. Thus $x$ is adjacent to at least two other vertices, contradicting $d(x) = 3$.

If $x$ is adjacent to one vertex of $P$ consider Fig. 4:

Since $\langle\{3,4,7,8,9,10\}\rangle$ is a 6-cycle, $x \sim a$, where $\{3,9,10\} \subset N(a)$, and $x \sim b$, where $\{4,7,8\} \subset N(b)$. Thus, $\langle\{x, a, b, 1, 2, 3, 4, 5, 7, 10\}\rangle = P'$ is a Petersen graph. Replace $P$ by $P'$ and proceed as in the first case.

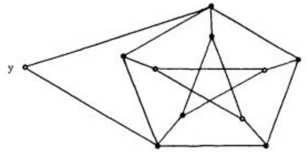


FIGURE 2.

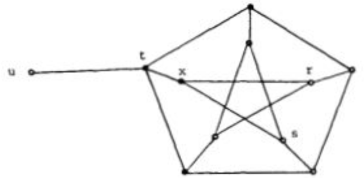


FIGURE 3.

Finally, assume $x$ has no neighbor in $P$. It is easy to show that $N(x) = \{x_1, x_2, x_3\}$, where $x_1, x_2, x_3 \in V_3$ (notation as in the first case). Now, $V(G) - V(P) = V_0 \cup V_1 \cup V_2 \cup V_3$, where

$$V_0 = \{y \in V(G) - V(P) \mid N(y) \cap V(P) = \emptyset\}.$$

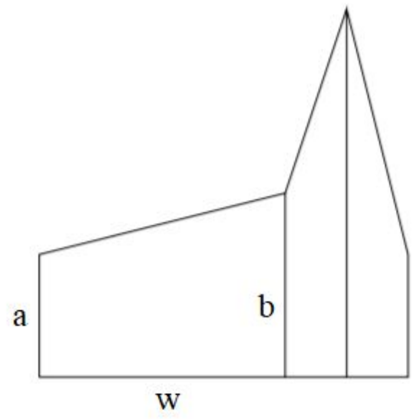
We use the same labeling procedure as in the first case with these additional rules: for $e = (y, z)$,

$$\omega_e(y) = 1,\ \omega_e(z) = 0 \qquad \text{if } y \in V_0,\ z \notin V_0,$$
$$\omega_e(y) = \omega_e(z) = 1/2 \qquad \text{if } y, z \in V_0.$$

It is straightforward, but tedious, to show that for all $u \in V(G) - V(P)$, $\omega(u) \geq 3$ *unless* $u \in V_1$ and $N(u) \cap (V_1 \cup V_2 \cup V_3) = \{a_1, a_2, a_3\}$ with $a_1, a_2, a_3 \in V_2$. However, if this happens $u \nsim x$ and $u \nsim x_i$ for $i = 1, 2, 3$; this contradicts $d(u, x) \leq 2$. ∎

## CCC S1

Loop over every fence piece and add (a+b) * w/2 to the answer.



Make sure you use a floating-point data type and to format output correctly

cout << fixed; in c++

## CCC S2

Flipping the colour of a row twice reverts it back to its original colour. Therefore, we only need to consider the number of times a row has been flipped mod 2.

Since MN <= 5e6, looping through all the cells after processing the colour changes suffices.

If we let row[i] represent the number of times the ith row has been flipped, and col[j] the number of times the jth column has been flipped..

A cell (i,j) is coloured gold if (row[i] + col[j]) mod 2 = 1.

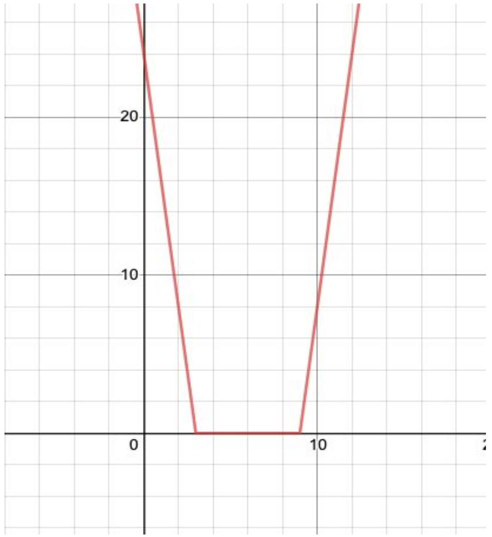
There exists a method of counting that doesn't require looping through all the cells at the end. This is left as an exercise for the reader.

## CCC S3

Person i starts at $p_i$, takes $w_i$ seconds to walk 1 meter, and can hear within $d_i$ meters.

We can represent the time it takes for them to reach the position x as a piecewise function $f_i(x)$.

$$f_i(x) = \begin{cases} w_i \cdot |x - (p_i - d_i)| & x < p_i - d_i \\ 0 & p_i - d_i \leq x \leq p_i + d_i \\ w_i \cdot |x - (p_i + d_i)| & p_i + d_i < x \end{cases}$$



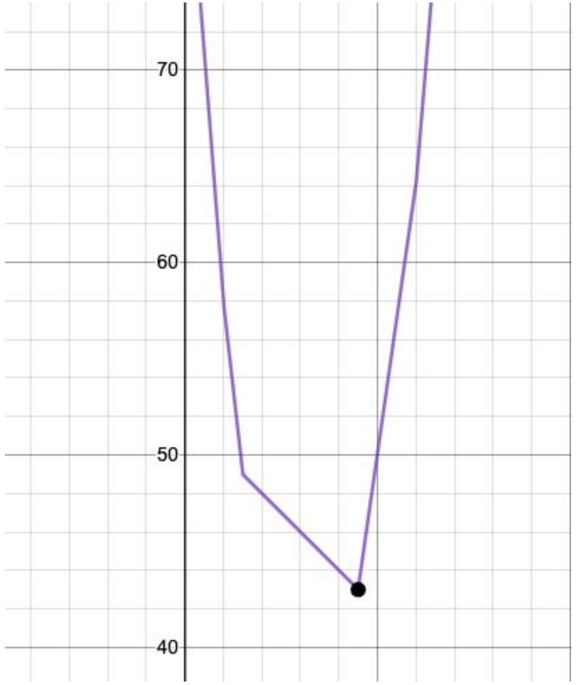
## CCC S3

We notice that each of these functions is a “convex” function.

Therefore, the total sum of all functions is also a convex function.

Finding the minimum value can be done using “ternary” search.

Alternatively you can use binary search or line sweep.



## CCC S4

Key points:

- Find the shortest way to get from station 1 to N for each day
- There is only one subway train
- Visits the stations in a defined order
- Two stations will switch orders each day
- There are walkways that we can take that stay the same

## CCC S4

Let us consider finding the shortest path for a given day before we consider the updates. The first observation is that it is never more optimal to return to the subway after getting off. There is only 1 train so you always have to wait for it to arrive at your current station before you can take it. This is the same as taking the subway from station 1.

The minimum time can then be represented as

$$\min\{a_1, \dots, a_N\}$$

where $a_i$ is the subway time from 1 to i + walking time from i to N

## CCC S4

With this observation the problem boils down to finding the shortest walkway path from the each station to station N as the time it takes to go from station 1 to i via subway is given based on the initial order of the stations.

We can use BFS to find SSSP to station N from each station. Then we can iterate over each station and find the minimum.

Now that we've found the minimum for one day, how does this translate to updates between days?

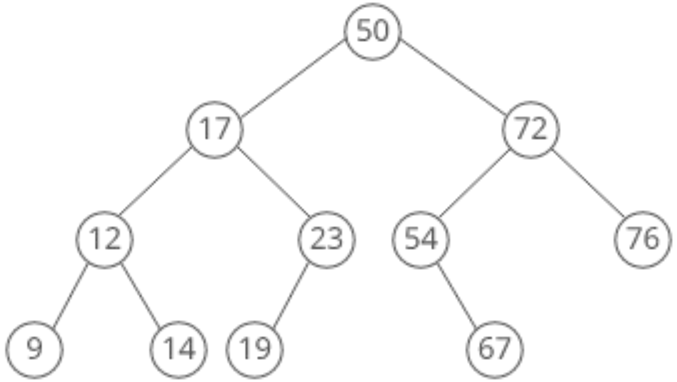
## CCC S4

The second observation to make is that only two times change when an update is made. Every other station still has the same position and the walkways remained unchanged.

After performing the two updates, we want to find the minimum time out of all of the stations without recalculating everything. One possible solution is with segment tree as we can update the time for two points while maintaining range minimum. A simpler implementation is using map/multiset.

## CCC S4

Maps/multisets allow us to maintain a sorted array while performing updates.



To perform an update, we remove the old times and add the two new ones. We can then lookup the first element in the map/multiset to find the minimum time after each update.

## CCC S5

Problem: construct an array that satisfies some requirements, $x_i$, $y_i$, $z_i$ means the gcd of all the elements in the subarray from $x_i$ to $y_i$ should equal $z_i$.

First Subtask: $1 <= z_i <= 2$.

We see that the elements covered by requirements with $z_i = 2$ should all be 2. The requirements with $z_i = 1$ need at least one 1 in the corresponding subarray. After placing all the 2s, we can check if the remaining constraints can be satisfied fairly simply.

## CCC S5

Full Solution: $1 <= z_i <= 16$.

For an element to be covered by ranges with gcds of $z_{i1}, z_{i2}, \dots, z_{ik}$ means it should be divisible by all of them. Since each element should be less than some threshold (1e9), the smallest element that is a multiple of all of these values is $\mathrm{lcm}(z_{i1}, z_{i2}, \dots, z_{ik})$.

We can notice that the constraints are still fairly small, $z_i <= 16$, so we can just maintain a "difference array".

After constructing our array, we still need to check it satisfies all of the requirements, one solution is constructing a sparse table that can query range gcds.

Alternatively, some people used a segment tree that can handle both range lcm updates and range gcd queries.