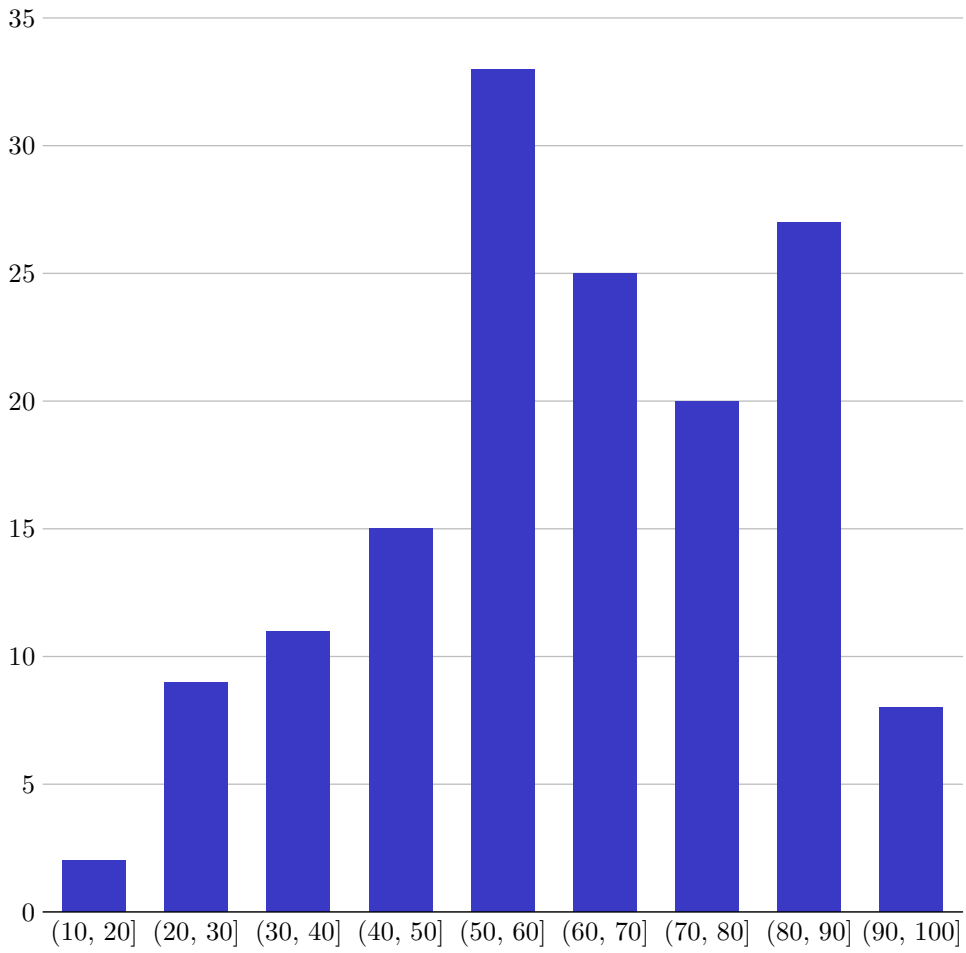
Quiz 2



Mean: 62.65; Median: 62.5; Standard deviation: 19.53

Problem 1. Bracelet Division [25 points]

A band of k thieves has stolen a bracelet adorned with t different types of jewels. They want to divide the bracelet fairly between them by cutting the bracelet in a small number of places and dividing the parts. For each type $i = 1, 2, \ldots t$, if there are c_i jewels of this type, then each thief wants at least $\lfloor c_i/k \rfloor$ of the jewels.

The thieves ask Professor X for help. The professor models the problem as a continuous problem: the open bracelet is the unit interval, and it is divided into sub-intervals that represent consecutive jewels of each type. The professor finds a way to partition the "continuous bracelet" fairly between the thieves using $(k-1)t$ cuts. However, those cuts do not necessarily correspond to legitimate cuts of the real bracelet, as they may cut across jewels, crushing them.

Given the cuts suggested by the professor, show how to efficiently compute $(k-1)t$ legitimate cuts that divide the actual jewel bracelet between the thieves to their satisfaction.

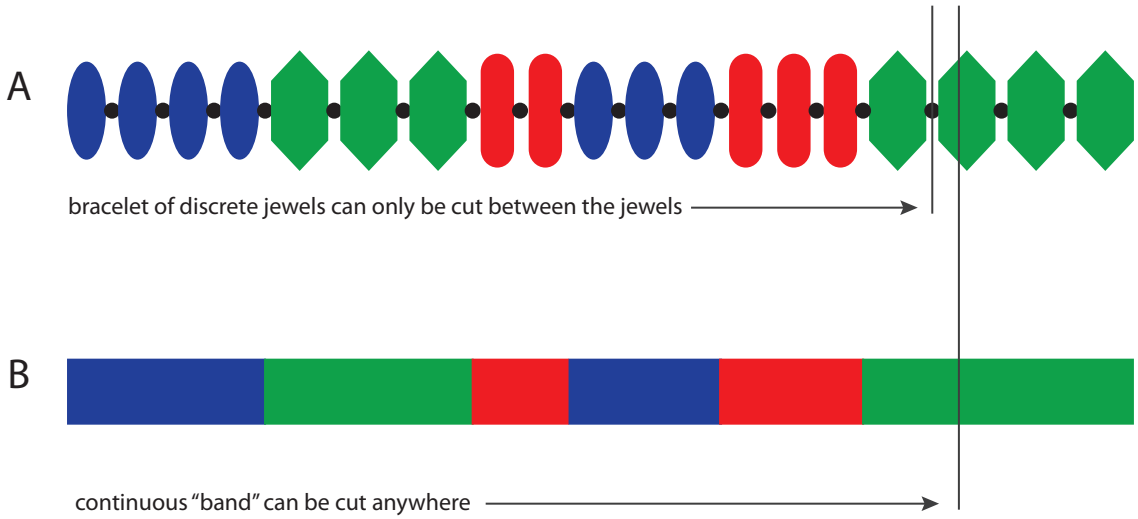


Figure 1: Example of (A) a bracelet with jewels and (B) the continuous version of the same bracelet. Notice how a cut across the continuous bracelet may crush a jewel.

Solution:

Executive Summary The most efficient approach for the bracelet division problem is to recast it in terms of a network flow problem and solve using the Ford-Fulkerson algorithm with overall runtime $O(t^2k^2)$.

Algorithm Let $s_{i,j}$ correspond to the respective uncut quantity of jewel of type i to thief j. All of the uncrushed jewels are distributed to the thieves according to the solution specified by the professor.

Now, all that is left is to distribute the crushed jewels between the thieves in an optimal manner. Let m be the number of crushed jewels. Let $r_{ij} = \max(\lfloor c_i/k \rfloor - s_{i,j}, 0)$ represent the residual quantity of jewel of type i that thief j needs for the fairness constraint to be fulfilled. The flow network for distributing the crushed jewels then is constructed as follows:

1. Create the following vertices in the network:

- source node S
- nodes u_i for $i \in \{1, 2, \ldots, m\}$ corresponding to the crushed jewels
- nodes v_{ij} for $i \in \{1, 2, \ldots, t\}$ and $j \in \{1, 2, \ldots, k\}$ corresponding to the pairing of jewel type i with thief j.
- sink node T

2. Create edges from

- S to each u_i with capacity $c(S, u_i) = 1$
- u_l (each crushed jewel) to $v_{i,j}$ where i is the type of the crushed jewel and j is a thief who can receive this jewel, with capacity $c(u_l, v_{i,j}) = 1$
- from node $v_{i,j}$ to sink node T with capacity $c(v_{i,j}, T) = r_{ij}$

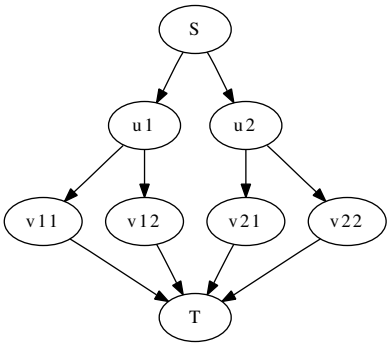


Figure 2: Illustration of the Network Flow Graph for a bracelet with 2 crushed jewels, 2 thieves, and 2 jewel types.

3. We then run Ford-Fulkerson on this network. The assignment of crushed jewels to thieves is determined by looking at the nonzero flows from the nodes $v_{i,j}$ to the sink node.

Correctness

Lemma 1 *There exists a valid division of jewels among thieves if and only if a max flow in G saturates the edges $(v_{i,j}, T)$ $\forall i \in \{1, \ldots, t\}$ and $j \in \{1, \ldots, k\}$.*

PROOF.

Consider a valid division of jewels among thieves. Define the flow f with values $\forall i \in \{1, \ldots, t\}, j \in \{1, \ldots, k\}, l \in \{1, \ldots, m\}$

$$f(u_l, v_{i,j}) = 1$$
$$f(v_{ij}, T) = r_{i,j}$$
$$f(S, u_l) = 1$$

f may not satisfy flow conservation if some thieves get more jewels than necessary. However, we can adjust the flow by zeroing out the flow of 1 on appropriate edges from the source to u_l and corresponding edges from u_l to v_{ij} until the incoming cumulative flow to every node v_{ij} is equal to the outgoing residual flow of r_{ij}. Clearly f still satisfies the capacity constraints, so f is a valid flow in G. By construction, f saturates the edges (v_{ij}, t) $\forall i, j$. Thus, $|f|$ equals the capacity of the cut $(s, V - s)$. By the min-cut max-flow theorem, f is a max flow.

For the converse, let f be a max flow in G that saturates the edges (v_{ij}, t) $\forall i \in \{1, \ldots, t\}, j \in \{1, \ldots, k\}$.

Suppose f is found through the Ford-Fulkerson method. By the integrality theorem (c.f. recitation or CLRS 3rd ed., Theorem 26.10), all of the flow values of f are integer. We can determine the appropriate assignment of jewels to thieves by looking at the saturated edges from u_l to $v_{i,j}$.

□

Lemma 2 *A max flow in G always saturates the edges (v_{ij}, T) $\forall i \{1, \ldots, t\}$ and $j \{1, \ldots, k\}$.*

PROOF. We can compute the residual $w_{ij} = c_i/k - s_{ij}$ of jewel of type i that needs to be allocated to thief j in the continuous version. Let z_{lij} be the fraction of the fractured jewel l of type i that was assigned to thief j by the professor.

Define the flow f with values $\forall i \in \{1, \ldots, t\}, j \in \{1, \ldots, k\}, l \in \{1, \ldots, m\}$

$$f(S, u_i) = 1$$
$$f(u_l, v_{ij}) = z_{lij}$$
$$f(v_{ij}, T) = w_{ij}$$

By construction the (v_{ij}, T) edges are saturated. We show that f is a valid flow. First, we note that it satisfies the capacity constraints. Conservation of flow may not be satisfied. So, we use the

approach in the previous lemma by reducing the flows from the source to u_l and from u_l to v_{ij} equally until the incoming flow to every node v_{ij} is equal to outgoing flow r_{ij}. The modified flow satisfies flow conservation and capacity constraints. $|f| = c(s, V - s)$, so f is a max flow.

□

The above lemma says we can always construct such a max flow, so it follows that there always exists a valid assignment of jewels to thieves.

Note that we did not take off points for the failure to address the scenario when a jewel is cut into more than 2 pieces.

Running Time Analysis Graph construction takes $O(kt)$ time. The running time of Ford-Fulkerson is $O(Ef^\star)$ where $f^\star$ is the maximum flow. The maximum flow is at most equal to the number of crushed jewels, so $f^\star \leq kt$. Furthermore, since there are $O(kt)$ edges in the graph, we obtain $O(k^2t^2)$ runtime.

Alternative Solutions Many students proposed a greedy, $O(kt)$ time solution to the problem, that rounds a jewel up or down based on the jewel type quantity already allocated to a particular thief. This solution only works under the assumption that jewels of the same type are continuous in the original bracelet, which, as indicated on the accompanying graph of the problem, is not necessarily true. We decided to give 20% for such solutions, and up to 30% if you state this assumption explicitly.

Problem 2. Dividing the Delta Quadrant [25 points]

In year 3512, the Star Federation has finally reached a consensus in the century long conflict between Treaps and Cycles for the asteroid field in the Delta Quadrant. To avoid any tension between the two factions, the decision has been made to divide the asteroids in a way that maximizes the minimum distance between any two asteroids assigned to different factions, even if the number of asteroids given to each is uneven (but no faction shall be left without any asteroids).

Design an efficient algorithm which, given a list of n asteroid coordinates (a 3-tuple of real numbers) in the Delta Quadrant, returns the optimal partition maximizing the minimum distance between any two asteroids assigned to different partitions.

Solution:

Executive Summary. We will convert this to a graph problem. We will then apply Prim's algorithm to find the MST in $O(n^2)$ worst case time. After finding the MST, we will pick out the heaviest edge and remove it to disconnect the MST into two parts, these parts will correspond to the assignment given to the two factions.

Algorithm. First we will make each asteroid a node in our graph. Then, we can compute the cartesian distance between any two asteroids and assign them as the weight of the edge between any pair of asteroids. $w = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$.

We then find the MST of the graph. This is a dense graph. After the MST is found, we remove the heaviest edge (break ties arbitrarily). Removing this edge disconnects the MST and divides the nodes into two parts. We can identify the content of each part via a BFS. We then return the two parts and assign them arbitrarily to the Treaps and the Cycles.

Correctness. First, consider any partition of the original graph into two groups C and $T = V - C$, this is a cut. Let $d(C, T)$ be the minimum distance between any two asteroids assigned to different factions, it is clear that $d(C, T)$ is equal to the weight of the lightest edge crossing the cut between C and T.

We will also need a lemma: Given a graph and any MST of the graph, the weight of the heaviest edge of the MST is always the same.

Proof by contradiction: Suppose the graph admits two MSTs T and T', such that T contains an edge that is heavier than all edges in T'. Consider the cut of the graph formed by removing the heaviest edge in T. Since T' is spanning, at least one edge of T', call it e', will cross the cut, which has a lighter weight than the edge we removed. This means that the heaviest edge we removed from MST T is not the lightest edge crossing the cut. Now consider adding e' to T. There will be a cycle with both e and e' in it. Thus, we can remove e and get a lighter weight for the MST, which contradicts the fact that T used to be an MST.

Now let's prove the original proposition by contradiction. Suppose the cuts returned by our algorithm is C^* and T^*, and the edge removed by our algorithm has weight M, which by the lemma is the unique weight of the heaviest edge in any MST of the graph. Suppose there exists a better partition C and T, such that $d(C,T) > d(C^*,T^*) = M$. Let the lightest edge crossing the cut C, T be e, then $w(e) = d(C,T) > d(C^*,T^*) = M$. By the MST property, e can be included in some MST of the graph. But on the other hand, $w(e) > M$, the unique largest weight of any edge in any MST of the graph. Contradiction.

Running Time. Our graph has n nodes and $O(n^2)$ edges. This is a dense graph, so Prim's algorithm will do better asymptotically than Kruskal's. In fact, Prim's algorithm runs in $O(E + V \lg V) = O(n^2)$ time, while Kruskal's algorithm runs in $O(E \lg V) = O(n^2 \lg n)$ time. To identify the contents of each part after the removal of the heaviest edge, we use BFS (starting with the two nodes of the removed edge) which takes $O(V + E) = O(n^2)$ time. Thus, the algorithm runs in $O(n^2)$ time.

Alternative Solutions. Many students used Kruskal's instead of Prim's to get a slightly worse run time of $O(n^2 \lg n)$, usually accompanied by a slightly different proof of correctness; those solutions received at most 90 points.

Many other students implemented their own algorithms which were in essence the same as Prim's or Kruskals's, but failed to realize the connection.

There is another class of solutions where you repeatedly take out the heaviest edge in the graph, until the graph is disconnected into two parts. This algorithm takes $O(n^4)$ time since there are $O(n^2)$ edges to remove, and checking wether the graph is connected each time using BFS or DFS takes $O(n^2)$ time. However, if you sort the weights of the graphs first and then do a binary search on the graphs, you can imporve the runtime of this method to $O(n^2 \lg n)$, which is the same as Kruskals. In the latter case, you will be awarded the same number of points as people who used Kruskal's, assuming everything else is correctly written.

A fairly common incorrect solution is as follows. Find the two points which are farthest apart, and assign them to T and C respectively, then build the two separate trees with something similar to Prim's algorithm until a point where another connection made will necesarrily connect the two groups. This is incorrect because you can come up with counter examples where the optimal assignment necessarily assigns the same color to the two farthest points. For example, consider a configuration in 2D where there is one astroid in position (0,1), and many many astroids densely populating the line from (-1,0) to (1,0). Clearly, the partition should be giving the one point to one faction, and give all the other points to another, but the two farthest points both belong to the same faction.

Another thing to note is that some students used Prim's algorithm but incorrectly stated that the runtime is amortized; those solutions lost a few points. This is probably due to the fact that Prim's algorithm uses Fibonacci Heap, which we mentioned have amortized runtime for its operations.

Thus, each single step of Prim's may be costly, but if we execute Prim's to completion, the total amount of work is worst-case $O(E + V \lg V)$.

Problem 3. A First World Problem [25 points]

Billy Billionaire wants to create an artificial lake in the mountains near his mansion. Fortunately, he lives in 2D land, which makes the problem much easier. He has elevation data, taken at regular intervals across the two-dimensional mountains. We can write this elevation data as $A[1], \ldots, A[n]$. The stretch of land from i to j can be made into a lake if and only if for all $i < k < j$, $A[k] < A[i]$ and $A[k] < A[j]$. We say that the height of the lake $[i, j]$ is equal to $\min\{A[i], A[j]\}$, and the width of the lake $[i, j]$ is equal to $(j - i - 1)$. See Figure 3 for an example.

Billy wants to pick the best stretch of land $[i, j]$ on which to build a lake. He wants the lake to be very wide, so that it looks impressive. However, the lake height should be relatively low, so that his fleet of helicopters have an easier time airlifting the water to the new lake. After some contemplation, Billy decides that he wants to pick a stretch of land $[i, j]$ that can be made into a lake that maximizes $(\text{lakewidth} - \frac{1}{100} \cdot \text{lakeheight})$. Design an efficient algorithm that finds a lake maximizing Billy's chosen objective function.

Solution:

The key property of the problem is that a particular column can be the lower bank for at most two lakes: one extending to the left and one extending to the right. This allows us to quickly find the $O(n)$ feasible lakes and then the one among them which maximizes Billy's objective.

Algorithm: Traverse the land from left to right. Use a stack to keep track at each column j of each column i, $i < j$ (to the left) such that for all columns k in between ($i < k < j$), we have that $A[k] < A[i]$ and $A[k] < A[j]$. Collect all such pairs (i, j) as feasible lake end points.

```
1  Lakes = SET()
2  Stack = STACK()
3  for j = 1 to n
4      lakeAllowed = TRUE
5      while not Stack.EMPTY()
6          i = Stack.TOP()
7          if lakeAllowed
8              Lakes.ADD((i, j))
9          if A[i] > A[j]
10             exit while loop
11         if A[i] = A[j]
12             lakeAllowed = FALSE // All subsequent lakes would end at current i.
13         Stack.POP()
14     Stack.PUSH(j)
```

Next, find the lake in $Lakes$ which minimizes Billy's objective by simple iteration.

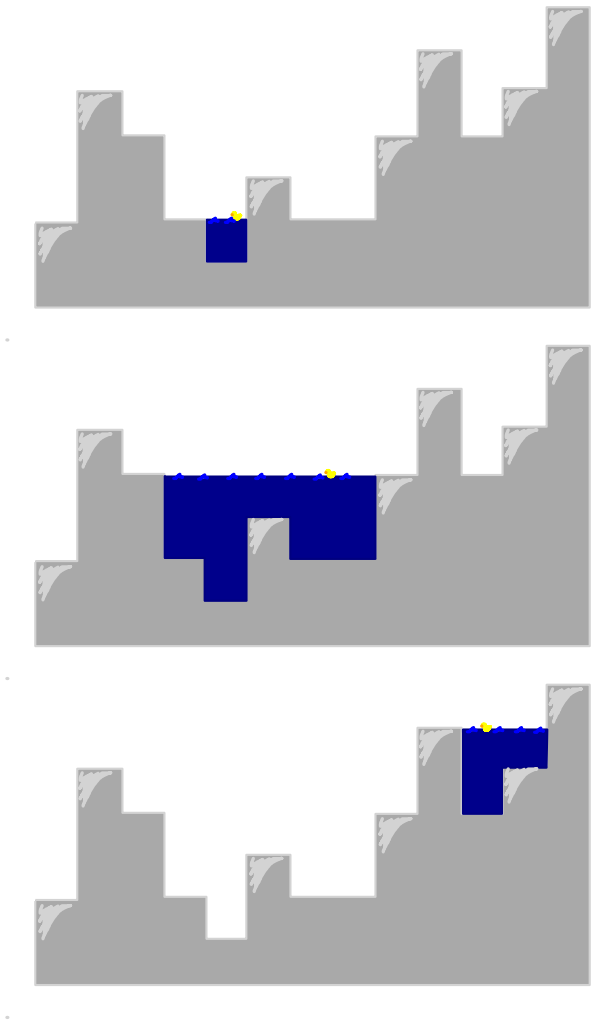


Figure 3: Three different possible lakes on the mountain region described by the array $A = [2, 5, 4, 2, 1, 3, 2, 2, 4, 6, 4, 5, 7]$. The first lake has $\text{lakewidth} = 1$ and $\text{lakeheight} = 2$; the second has $\text{lakewidth} = 5$ and $\text{lakeheight} = 4$; the third $\text{lakewidth} = 2$ and $\text{lakeheight} = 6$.

Correctness: Note that all columns $1 \ldots n$ are pushed onto the stack in order (line 14). Therefore, within the while loop (lines 5-13), $i < j$, and i is strictly decreasing. Also, for any k such that $i < k < j$, k needs to be popped from the stack before (i, j) is considered in line 8. Furthermore, column i is popped from the stack iff a column j such that $i < j$ and $A[i] \leq A[j]$ is subsequently pushed onto the stack. We will refer to this as "i *is popped by* j", even if i is popped by some $k < j$.

We show that the lakes considered for Billy's objective are all and only the feasible lakes.

"$\Rightarrow$" All the pairs (i, j) considered in line 8, are such that for all k, $i < k < j$, we have $A[k] < A[i]$ and $A[k] < A[j]$. Otherwise, if $A[k] > A[j]$, k would not be popped by j, if $A[k] = A[j]$, $lakeAllowed$ would be false for the remainder of the **while** loop, and if $A[k] \geq A[i]$, i would be popped by k. In all cases, (i, j) would not be considered.

"$\Leftarrow$" If a pair (i, j) is feasible, it will be considered in line 8. Otherwise, either there exists k, $i < k < j$ such that $A[k] > A[j]$ which prevents j from popping k and reaching i; i is popped from the stack in an earlier iteration of the **for** loop, before j is pushed, so there exists k, $i < k < j$ such that $A[k] \geq A[i]$; or $lakeAllowed$ is true which means there exists k, $i < k < j$ such that $A[k] = A[j]$. All cases contradict feasibility.

Running Time: Although multiple columns could be popped from the stack in one iteration of j, each column of the land is pushed onto the stack exactly once and popped at most once. Therefore, the total work done is $O(n)$ to generate $Lakes$, and $O(n)$ to find the best lake.

Alternative Solutions: Note that for correctness it is essential that all feasible lakes are considered, whether $A[i] \leq A[j]$ or $A[i] \geq A[j]$. Some implementations would miss lakes with a taller bank on one side or the other, for instance, by considering candidate lakes only within a "**while** $A[i] < A[j]$" loop.

To avoid this mistake (and loss of credit), many correct solutions traverse the land twice, once in each direction, first focusing on lakes $i < j$ such that $A[i] \leq A[j]$ then $A[i] \geq A[j]$.

A handful solutions proposed a more complex way of enumerating the feasible lakes in $O(n)$ time: Find the local minima in land (pits), then progressively grow the first one by raising the water level. When it spills over, move to the next minimum, and repeat, merging them as needed. This method requires more bookkeeping, but still received full credit if there were no errors.

Correct solutions which enumerate feasible lakes in $O(n \lg n)$ time or slower received partial credit.

Problem 4. Social Networking [25 points]

MITbook+ is a new social networking site created by MIT students. Much like other social networking sites, MITbook+ allows you to establish "friendship" with other users. Each user has a unique MITbook+ ID number, and a list of all MITbook+ ID numbers is available to the general public. Furthermore, for any pair of user IDs, there is a publically-accessible webpage that can be used to check whether the pair are friends. However, in the interests of preserving user privacy, all other information associated with MITbook+ IDs is available to members only.

Currently, you are not a user of MITbook+, but you want to learn more about a specific person (known to you by his ID as 60461337). Specifically, you want to determine whether he went to school at MIT. Fortunately, the founders of MITbook+ have posted some statistics on their blog that may help you. Currently, MITbook+ has N users (a very large number). Exactly $2N/3$ users attended MIT; the other $N/3$ did not. Interestingly, because MIT is a very tight-knit community, every current or former MIT student on MITbook+ is friends with every other current or former MIT student on MITbook+.

(a) In the blog post with the statistics, the MITbook+ founders originally claimed that every user on MITbook+ who did not attend MIT was friends with at most $N/4$ users who did attend MIT. Create an efficient algorithm that uses this fact to determine whether user 60461337 attended MIT.

Solution:

Executive Summary. We use a randomized algorithm that samples $O(1)$ users to approximate the number of friends the student has. If 60461337 has sufficiently many friends, he must have attended MIT; otherwise, he could not have attended MIT. This results in an algorithm with runtime $O(1)$ and probability of correctness at least $\frac{2}{3}$.

Algorithm. Create a variable ℓ which is initialized to 0. Repeat the following operation $k = 2304$ times: pick a random user ID x and increment ℓ by 1 if x is friends with 60461337. After this process, if $\ell \geq \frac{15}{24} \cdot k$, return "$60461337$ is an MIT student." Otherwise, return "60461337 is not an MIT student."

Correctness. For simplicity of notation, let $z = 60461337$. Let U be the set of all user IDs, let M be the set of all MIT students, and let $A = U - M$ be the set of all other students. Let $f(x, y)$ be a boolean relation that is true if and only if x and y are friends. Then we can use the following notation to count the number of friends that a given user x has in a particular set S:

$$g_S(x) = |\{y \in S \mid f(x, y) = \text{TRUE}\}|$$

This notation lets us rewrite the facts that we are given as follows:

1. The number of users is $|U| = N$.
2. The number of users who attended MIT is $|M| = 2N/3$. The number of users who did not attend MIT is $|A| = N/3$.
3. Every pair of MIT students is friends: $\forall x \neq y \in M : f(x, y) = \text{TRUE}$. This further means that for any MIT student $x \in M$, the number of friends $g_M(x)$ that he has at MIT is $|M| - 1 = 2N/3 - 1$ (every MIT user but himself).
4. Every user $x \in A$ who did not attend MIT is friends with at most $N/4$ MIT students: $g_M(x) \leq N/4$.

We consider two cases:

- **Case 1:** The student z did not attend MIT. Then he can have at most $N/3 - 1$ non-MIT friends, and at most $N/4$ MIT friends. More formally:

$$g_U(z) = g_M(z) + g_A(z) \leq \frac{N}{4} + \frac{N}{3} - 1 = \frac{7}{12} \cdot N - 1$$

  In this case, the algorithm will return the incorrect answer if $\ell \geq \frac{15}{24} \cdot k$. What is the probability of that occurring? To compute this, we resort to the Chernoff bound.

  Let X_i be an indicator random variable that is equal to 1 if the ith random user is friends with z, and 0 otherwise. For simplicity, assume that R is selected with replacement — that we may select a given person more than once. So the probability of X_i being equal to 1 is the same as the probability of selecting one of z's friends:

$$p = \Pr[X_i = 1] = \frac{g_U(z)}{N} \leq \frac{\frac{7}{12} \cdot N - 1}{N} \leq \frac{7}{12}$$

  The computed value ℓ is the sum of the X_is, so we know that it's distributed according to the binomial distribution. We also know that $\mathbb{E}[\ell] = \frac{7}{12} \cdot k$. Therefore, we can use the Chernoff bound as follows:

$$\begin{aligned}
\Pr\left[\ell \geq \frac{15}{24} \cdot k\right] &= \Pr\left[\ell \geq \mathbb{E}[\ell] + \left(\frac{15}{24} \cdot k - \mathbb{E}[\ell]\right)\right] \\
&= \Pr\left[\ell \geq \mathbb{E}[\ell] + k \cdot \left(\tfrac{15}{24} - p\right)\right] \\
&\leq \exp\left(-\frac{k^2 \cdot \left(\frac{15}{24} - p\right)^2}{2k}\right) = \exp\left(-\frac{k}{2} \cdot \left(\tfrac{15}{24} - p\right)^2\right)
\end{aligned}$$

  Because p is strictly less than $\frac{15}{24}$, this bound on the probability increases with p. So the bound will still hold if we replace p with the largest value it could possibly take on:

$$\Pr\left[\ell \geq \frac{15}{24} \cdot k\right] \leq \exp\left(-k \cdot \tfrac{1}{2}\left(\tfrac{15}{24} - \tfrac{7}{12}\right)^2\right) = \exp\left(-\tfrac{k}{1152}\right)$$

  Because we picked $k = 2304$, this means that our probability of failure is no more than approximately 0.14. So our probability of success is at least $\frac{2}{3}$.

- **Case 2:** The student z attended MIT. Then he must have at least $2N/3 - 1$ friends. More formally:

$$g_U(z) = g_M(z) + g_A(z) \geq g_M(z) = \frac{2}{3} \cdot N - 1$$

In this case, the algorithm will return the incorrect answer if $\ell < \frac{15}{24} \cdot k$. What is the probability of that occurring? Well, the Chernoff bound will only let us bound the right tail of the distribution, so we need to do this somewhat differently than the first case. Define the indicator random variable Y_i to be 1 if and only if the ith random user is *not* friends with z. Then the probability of getting $Y_i = 1$ is equal to the probability of selecting a non-friend of z:

$$q = \Pr[Y_i = 1] = \frac{N - 1 - g_U(z)}{N} \leq \frac{N - 1 - \left(\frac{2}{3}N - 1\right)}{N} = \frac{N}{3N} = \frac{1}{3}$$

This means that we have a random variable $\sum Y_i = k - \ell$ that is distributed according to the binomial distribution, with probability q of success for each trial. This means that $\mathbb{E}[k - \ell] = qk$. This lets us use the Chernoff bounds as follows:

$$\begin{aligned}
\Pr\left[\ell < \frac{15}{24} \cdot k\right] &= \Pr\left[k - \ell > k - \frac{15}{24} \cdot k\right] \\
&\leq \Pr\left[k - \ell \geq \frac{9}{24} \cdot k\right] \\
&= \Pr\left[k - \ell \geq \mathbb{E}[k - \ell] + k\left(\tfrac{9}{24} - q\right)\right] \\
&\leq \exp\left(-\frac{k^2\left(\frac{9}{24} - q\right)^2}{2k}\right) \leq \exp\left(-k \cdot \tfrac{1}{2} \cdot \left(\tfrac{9}{24} - q\right)^2\right)
\end{aligned}$$

Again, this is maximized when q is largest, so we can safely bound this by substituting the largest value for q:

$$\Pr\left[\ell < \tfrac{15}{24} \cdot k\right] \leq \exp\left(-k \cdot \tfrac{1}{2} \cdot \left(\tfrac{9}{24} - \tfrac{1}{3}\right)^2\right) = \exp\left(-\tfrac{k}{1152}\right)$$

Again, our choice of $k = 2304$ ensures that this probability is less than a third. So our probability of correctness is at least $\frac{2}{3}$.

In summation, we have an algorithm that returns the correct answer with probability greater than $\frac{2}{3}$ no matter what our input is. As a result, we have a good randomized algorithm for solving this problem.

Running Time. We sample $k = 2304$ student IDs from the list of student IDs. This is equivalent to generating k random numbers between 1 and N, which can be done in time $O(k) = O(1)$. For each randomly selected user, we query the MITbook+ database (an operation which should take $O(1)$ time per query), and then decide whether 60461337 is an MIT student based on how many of the randomly sampled students were friends with 60461337. In total, this requires $\Theta(k) = \Theta(1)$ time.

Alternative Solutions. The solution given above was worth a maximum of 40% out of 40%. Many students gave a $\Theta(N)$ non-randomized solution counting the number of friends exactly, which was worth a maximum of 30% out of 40%.

(b) Not long after the original blog post, the founders updated their numbers. Apparently, the original claim spurred a large number of non-MIT users to friend more MIT students. Now, after this friending spree, every user on MITbook+ who did not attend MIT is friends with at most $2N/5$ users who did attend MIT. This also had the effect of raising the average number of friends per user to exactly $2N/3$. Given this updated information, devise an efficient algorithm that checks whether user 60461337 went to school at MIT.

Solution:

Executive Summary. We sample four users at random until we find four who are all friends. They are more likely than not to all be MIT students, so we check whether they are friends with 60461337 to determine whether 60461337 is also an MIT student. This algorithm has runtime $O(1)$ and correctness probability at least 0.51.

Algorithm. Repeat the following $k = 25$ times:

1. Pick four users a, b, c, d uniformly at random.
2. Check whether all four are friends. If not, continue to the next repetition of the loop.
3. Otherwise, check whether all four are friends with 60461337. If so, return "60461337 is an MIT student." If not, return "60461337 is not an MIT student."

If we finish k iterations without finding four friends, return "FAILED."

Correctness. For simplicity, we define a number of events:

1. Let F_{ab} be the event that a and b are friends.
2. Let F_{cd} be the event that c and d are friends.
3. Let F_{all} be the event that all four random users are all friends.
4. Let M_{ab} be the event that both a and b attended MIT.
5. Let M_{cd} be the event that both c and d attended MIT.
6. Let M_{all} be the event that all four random users attended MIT.

In order to correctly analyze the probability of correctness, we need to do some calculations. The average number of friends per user is $2N/3$. Therefore, the total number of friendships is $N \cdot (2N/3)/2 = N^2/3$. The number of friendships involving only MIT students is $(2N/3) \cdot (2N/3 - 1)/2 = 2N^2/9 - N/3$. This means that the number of friendships involving at least one user not from MIT is $N^2/3 - (2N^2/9 - N/3) = N^2/9 + N/3$. In order for a and b to be friends when they are not both from MIT, they must be part of just such a friendship. Hence:

$$\Pr[F_{ab} \wedge \neg M_{ab}] = 2 \cdot \frac{N^2/9 + N/3}{N^2} = \frac{2}{9} + \frac{2}{3N}$$

By symmetry, $\Pr[F_{cd} \wedge \neg M_{cd}] = \Pr[F_{ab} \wedge \neg M_{ab}]$. We can use this information to analyze cases as follows:

- **Case 1:** We want to figure out $\Pr[F_{all} \wedge M_{ab} \wedge M_{cd}]$. This is equal to the probability that a, b, c, and d are all unique MIT students. The probability of this is:

  $$\begin{aligned}\Pr[F_{all} \wedge M_{ab} \wedge M_{cd}] &= \tfrac{2N/3}{N} \cdot \tfrac{2N/3-1}{N} \cdot \tfrac{2N/3-2}{N} \cdot \tfrac{2N/3-3}{N} \\ &= \tfrac{2}{3}\left(\tfrac{2}{3} - \tfrac{1}{N}\right)\left(\tfrac{2}{3} - \tfrac{2}{N}\right)\left(\tfrac{2}{3} - \tfrac{3}{N}\right)\end{aligned}$$

  Because we know N is large, we can safely assume that $N > 1000$. Therefore, we can lower-bound the probability by 0.195.

- **Case 2:** We want to figure out $\Pr[F_{all} \wedge \neg M_{ab} \wedge M_{cd}]$. Because F_{all} can only occur when F_{ab} is true, we get the following equivalence:

  $$\begin{aligned}\Pr[F_{all} \wedge \neg M_{ab} \wedge M_{cd}] &= \Pr[F_{all} \wedge M_{cd} \mid F_{ab} \wedge \neg M_{ab}] \cdot \Pr[F_{ab} \wedge \neg M_{ab}] \\ &= \Pr[F_{all} \wedge M_{cd} \mid F_{ab} \wedge \neg M_{ab}] \cdot \left(\tfrac{2}{9} + \tfrac{2}{3N}\right)\end{aligned}$$

  At least one of a or b did not attend MIT. So the number of MIT students who could be friends with both a and b is at most $2N/5$. Hence, the probability of $F_{all} \wedge M_{cd}$ is at most the probability that c and d are among those $2N/5$ MIT students. This gives us the following upper-bound on the probability:

  $$\begin{aligned}\Pr[F_{all} \wedge \neg M_{ab} \wedge M_{cd}] &= \Pr[F_{all} \wedge M_{cd} \mid F_{ab} \wedge \neg M_{ab}] \cdot \left(\tfrac{2}{9} + \tfrac{2}{3N}\right) \\ &\leq \left(\frac{2N/5}{N} \cdot \frac{2N/5}{N}\right) \cdot \left(\frac{2}{9} + \frac{2}{3N}\right) \\ &= \frac{4}{25} \cdot \left(\frac{2}{9} + \frac{2}{3N}\right)\end{aligned}$$

  We know that N is very large, so we can safely assume that $N > 1000$. This means that we can upper-bound the probability of this event by 0.036.

- **Case 3:** We want to figure out $\Pr[F_{all} \wedge M_{ab} \wedge \neg M_{cd}]$. By the same analysis as in Case 2, the probability is at most 0.036.

- **Case 4:** We want to figure out $\Pr[F_{all} \wedge \neg M_{ab} \wedge \neg M_{cd}]$. We can upper-bound this probability as follows:

$$\begin{aligned}\Pr[F_{all} \wedge \neg M_{ab} \wedge \neg M_{cd}] &\leq \Pr[F_{ab} \wedge F_{cd} \wedge \neg M_{ab} \wedge \neg M_{cd}] \\ &= \Pr[F_{ab} \wedge \neg M_{ab}] \cdot \Pr[F_{cd} \wedge \neg M_{cd}] \\ &= \left(\tfrac{2}{9} + \tfrac{2}{3N}\right)^2\end{aligned}$$

  We know that N is very large, so we can safely assume that $N > 1000$. This means that we can upper-bound the probability of this event by 0.050.

First, let us analyze the probability that our algorithm will return "FAILURE." This can only occur if we never find four friends a, b, c, and d, even after trying $k = 25$ times. From the above, we know that the probability of finding four friends is at least 0.195. Therefore, the probability of failure in a single iteration is at most $1 - 0.195 = 0.805$. The probability of failing in all $k = 25$ iterations is at most $0.805^k \leq 0.01$. This contributes at most 0.01 to our probability of returning the wrong answer.

Now let's consider the probability of getting the right answer when we do find four people who are all friends with each other. Let $p = \Pr[F_{all} \wedge M_{all}]$, and let $q = \Pr[F_{all} \wedge \neg M_{all}]$. Then from Case 1, we know that $p \geq 0.195$. From Cases 2, 3, and 4, we know that $q \leq 2 \cdot 0.036 + 0.050 = 0.122$.

$$\Pr[M_{all} \mid F_{all}] = \frac{\Pr[M_{all} \wedge F_{all}]}{\Pr[F_{all}]} = \frac{p}{p+q} = \frac{1}{\frac{p+q}{p}} = \frac{1}{1 + \frac{q}{p}}$$

From the bounds on q and p above, we know that $\frac{q}{p} \leq 0.626$, and therefore that $\Pr[M_{all} \mid F_{all}] \geq 0.615$.

If 60461337 attends MIT, then our algorithm will return the right answer in the event that we end up with users a, b, c, d who all attended MIT. So the algorithm has two failure cases: returning "FAILURE," and picking four friends who are not all from MIT. Hence, the probability of failure when 60461337 is an MIT student is at most:

$$0.01 + \Pr[\neg M_{all} \mid F_{all}] = 0.01 + (1 - \Pr[M_{all} \mid F_{all}]) \leq 0.01 + 1 - 0.615 = 0.395$$

So our algorithm will be right more than half the time.

Now we consider the case where 60461337 does not attend MIT. Suppose we happen to pick a, b, c, d such that all were friends with 60461337. But if 60461337 is not an MIT student, then he is friends with at most $2N/5$ MIT students. So the probability of this failure occurring given that a, b, c, d are all MIT students is at most:

$$\frac{2N/5}{2N/3} \cdot \frac{2N/5-1}{2N/3-1} \cdot \frac{2N/5-2}{2N/3-2} \cdot \frac{2N/5-3}{2N/3-3} \leq \left(\frac{2N/5}{2N/3}\right)^4 \leq 0.130.$$

All told, there are three ways our algorithm could fail in the event that 60461337 is not an MIT student: returning "FAILURE," picking four friends who are not all from MIT, or picking four friends from MIT who happen to be friends with 60461337. More specifically, the probability of failure is at most:

$$0.01 + \frac{q + 0.130p}{p+q} = 1.01 - 0.870 \cdot \frac{p}{p+q} = 1.01 - 0.870 \Pr[M_{all} \mid F_{all}] \leq 0.475$$

Hence, our probability of success is again ≥ 0.51.

Running Time. We sample four random users a total of $k = 25$ times. We then check whether all four are friends with each other, requiring us to access the website $O(1)$ times. If we find four such friends, we then access the website another 4 tiems, to check whether all four are friends with 60461337. This is a constant number of operations.

Alternative Solutions. There are a number of deterministic $\Theta(N^2)$ algorithms for solving this problem. Many are based around the density of the friend network of 60461337: if he has between $2N/3$ and $11N/15$ friends, then the number of friendships between friends of 60461337 is much higher when 60461337 is an MIT student. A completely correct algorithm with runtime $\Theta(N^2)$ could get at most 40% out of the 60% allocated to this subproblem.

Some of these $\Theta(N^2)$ algorithms can be randomized to create $\Theta(N)$ or $\Theta(1)$ algorithms, although this greatly increases the difficulty of analysis. A correct algorithm with runtime $\Theta(1)$ could get as much as 60% out of 60% if written up well. A correct algorithm with runtime $\Theta(N)$ was worth at most 50% out of 60%.

One common mistake was an attempt to reuse the algorithm from part (a) with only a minor modification, such as returning a random answer when 60461337 has between $2N/3$ and $11N/15$ friends. However, when we analyze the correctness of an algorithm, we cannot look at the correctness of an average-case input — instead, we must examine the correctness of the worst-case input. The worst case input might be an MIT student with only $2N/3 - 1$ friends, or a non-MIT user with $11N/15 - 1$ friends. Hence, when the number of friends is ambiguous, it's not sufficient to flip a coin to decide whether 60461337 attends MIT. Algorithms that were mostly unchanged from part (a) were worth at most 10% of 60%.

6.046J / 18.410J Design and Analysis of Algorithms
Spring 2012