Please complete the survey:
https://goo.gl/forms/POfpZAPRWQcsxrWp2
Early Winter
Author: Tony Cai

- Just do it.
• Just do it.
• What else is there to say?
• Just do it.
• What else is there to say?
• Do not print anything besides the answer.
  e.g. print(“Please input n and $d_m$”).
Early Winter
Author: Tony Cai

- Just do it.
- What else is there to say?
- Do not print anything besides the answer.
  e.g. print("Please input n and \(d_m\)).
- Statistics; 59 solves / 126 attempted
Eating Out
Author: Tony Cai

Problem
Given $m$ objects, assign $a$, $b$, and $c$ objects to person 1, 2, and 3 respectively such that no object is assigned to all 3 people.

Statistics
51 solves / 254 attempted

Solution
Possible iff $a + b + c \leq 2 \cdot m$
Problem
Calculate the minimum amount of time a moving point is outside a circle

Statistics
20 solves / 192 attempted

Solution
Case analysis:
- Safety zone may stop shrinking before Anthony is in danger
- Anthony may be in danger and catch up to safety zone
- ...
Problem
Simulate a card game where on a player’s turn, she either gets knocked out or gets another life.

Statistics
12 solves / 139 attempted
Problem
Simulate a card game where on a player’s turn, she either gets knocked out or gets another life.

Solution
Suppose $k$ players are active, the current player is $p$, the current turn number is $t_1$, and the next turn number is $t_2$. The next player to draw a card is $(p + t_2 - t_1) \mod k$. 
Exploding Kittens

Problem
Simulate a card game where on a player’s turn, she either gets knocked out or gets another life.

Solution
Keep track of active players in an array, and update the array when a player is knocked out.

Time Complexity: $O(n^2 + |E| + |D|)$
Problem
Given strings $s, s_1, s_2$, check if $s$ can be partitioned into sub-sequences $s_1$ and $s_2$.

Statistics
17 solves / 197 attempted
Dynamic programming. Similar to the classical problem longest common sub-sequence (LCS).
Dynamic programming. Similar to the classical problem longest common sub-sequence (LCS).

Let $f(i, j)$ return whether it is possible to partition $s[i + j :]$ into $a[i :]$ and $b[j :]$. 
Dynamic programming. Similar to the classical problem longest common sub-sequence (LCS).
Let $f(i, j)$ return whether it is possible to partition $s[i + j :]$ into $a[i :]$ and $b[j :]$.  
Base case $f(|s_1|, |s_2|) = True$. Want to compute $f(0, 0)$. 
Homework

Solution

- Dynamic programming. Similar to the classical problem longest common sub-sequence (LCS).
- Let $f(i, j)$ return whether it is possible to partition $s[i + j :]$ into $a[i :]$ and $b[j :]$.
- Base case $f(|s_1|, |s_2|) = True$. Want to compute $f(0, 0)$.
- Recurrence relation is as follows:

$$f(i, j) = (f(i + 1, j) \land s[i + j] = s_1[i])$$
$$= \lor (f(i, j + 1) \land s[i + j] = s_2[j])$$.
Problem

Find a path between $s$ and $t$ in a graph that maximizes the minimum distance between a set of vertices and any vertex on the path. The length of the path is also constrained.
Solution

- Complex graph problem involving multiple algorithms in multiple steps. As a high level overview, the intended solution mainly uses Dijkstra’s SSSP and binary search.
Solution

- Complex graph problem involving multiple algorithms in multiple steps. As a high level overview, the intended solution mainly uses Dijkstra’s SSSP and binary search.

- First of all, for every node, compute its min distance to any spider. Sounds difficult, but is not any harder than Dijkstra’s. Imagine there’s only one spider/source, this step is easy for anyone who can implement Dijkstra’s. When there are multiple spiders/sources, simply push them all into heap in the beginning and mark their distances to be 0. The rest is identical to normal Dijkstra’s.
Solution

Complex graph problem involving multiple algorithms in multiple steps. As a high level overview, the intended solution mainly uses Dijkstra’s SSSP and binary search.

First of all, for every node, compute its min distance to any spider. Sounds difficult, but is not any harder than Dijkstra’s. Imagine there’s only one spider/source, this step is easy for anyone who can implement Dijkstra’s. When there are multiple spiders/sources, simply push them all into heap in the beginning and mark their distances to be 0. The rest is identical to normal Dijkstra’s.
For a vertex $v$, we'll call the min distance from $v$ to any spider $s(v)$.

Once $s(v)$ is known for every $v$, there are 2 likely scenarios.

1. Anthony is trying to avoid spiders too much, i.e. avoiding all and only vertices $v$ such that $s(v) < K$ for some constant $K$, however, this results in Anthony avoiding too many spiders and not making it in time.

2. Anthony is staying too close to spiders, i.e. Anthony's avoiding all and only vertices $v$ such that $s(v) < K$ for some constant $K$, however, Anthony could be avoiding more vertices than he is in order to increase his min distance to any spider, yet still making it in time.
Solution

- For a vertex $v$, We’ll call the min distance from $v$ to any spider $s(v)$.
- Once $s(v)$ is known for every $v$, there are 2 likely scenarios.
Solution

- For a vertex $v$, We’ll call the min distance from $v$ to any spider $s(v)$.
- Once $s(v)$ is known for every $v$, there are 2 likely scenarios.
- 1. Anthony is trying to avoid spiders too much, i.e. avoiding all and only vertices $v$ such that $s(v) < K$ for some constant $K$, however, this results in Anthony avoiding too many spiders and not making it in time.
Solution

- For a vertex $v$, we'll call the min distance from $v$ to any spider $s(v)$.
- Once $s(v)$ is known for every $v$, there are 2 likely scenarios.
  1. Anthony is trying to avoid spiders too much, i.e. avoiding all and only vertices $v$ such that $s(v) < K$ for some constant $K$, however, this results in Anthony avoiding too many spiders and not making it in time.
  2. Anthony is staying too close to spiders, i.e. Anthony’s avoiding all and only vertices $v$ such that $s(v) < K$ for some constant $K$, however, Anthony could be avoiding more vertices than he is in order to increase his min distance to any spider, yet still making it in time.
Solution

- For a vertex \(v\), we'll call the min distance from \(v\) to any spider \(s(v)\).
- Once \(s(v)\) is known for every \(v\), there are 2 likely scenarios.
  1. Anthony is trying to avoid spiders too much, i.e. avoiding all and only vertices \(v\) such that \(s(v) < K\) for some constant \(K\), however, this results in Anthony avoiding too many spiders and not making it in time.
  2. Anthony is staying too close to spiders, i.e. Anthony’s avoiding all and only vertices \(v\) such that \(s(v) < K\) for some constant \(K\), however, Anthony could be avoiding more vertices than he is in order to increase his min distance to any spider, yet still making it in time.
Solution

3. There is a third scenario. Anthony’s avoiding all and only vertices \( v \) such that \( s(v) < K \) for some constant \( K \), such that

\( a. \) if Anthony avoids \( v \) such that \( s(v) < K + 1 \), this results in scenario 1. where Anthony avoids too many spiders. i.e. \( K \) is too large.

\( b. \) if Anthony avoids \( v \) such that \( s(v) < K - 1 \), this results in scenario 2. where Anthony avoids too few spiders. i.e. \( K \) is too little.
3. There is a third scenario. Anthony’s avoiding all and only vertices $v$ such that $s(v) < K$ for some constant $K$, such that

a. if Anthony avoids $v$ such that $s(v) < K + 1$, this results in scenario 1. where Anthony avoids too many spiders. i.e. $K$ is too large.

b. if Anthony avoids $v$ such that $s(v) < K − 1$, this results in scenario 2. where Anthony avoids too few spiders. i.e. $K$ is too little.

Binary search for $K$.

For each $K$, use normal Dijkstra’s from $s$ to $t$ on the sub-graph, where vertices $v$ such that $s(v) < K$ are ignored. The failure condition is if Anthony does not make it in time.
3. There is a third scenario. Anthony’s avoiding all and only vertices \( v \) such that \( s(v) < K \) for some constant \( K \), such that

a. if Anthony avoids \( v \) such that \( s(v) < K + 1 \), this results in scenario 1. where Anthony avoids too many spiders. i.e. \( K \) is too large.

b. if Anthony avoids \( v \) such that \( s(v) < K - 1 \), this results in scenario 2. where Anthony avoids too few spiders. i.e. \( K \) is too little.

Binary search for \( K \).
Solution

3. There is a third scenario. Anthony’s avoiding all and only vertices $v$ such that $s(v) < K$ for some constant $K$, such that

a. if Anthony avoids $v$ such that $s(v) < K + 1$, this results in scenario 1. where Anthony avoids too many spiders. i.e. $K$ is too large.

b. if Anthony avoids $v$ such that $s(v) < K - 1$, this results in scenario 2. where Anthony avoids too few spiders. i.e. $K$ is too little.

Binary search for $K$.

For each $K$, use normal Dijkstra’s from $s$ to $t$ on the sub-graph, where vertices $v$ such that $s(v) < K$ are ignored. The failure condition is if Anthony does not make it in time.
3. There is a third scenario. Anthony’s avoiding all and only vertices $v$ such that $s(v) < K$ for some constant $K$, such that

a. if Anthony avoids $v$ such that $s(v) < K + 1$, this results in scenario 1. where Anthony avoids too many spiders. i.e. $K$ is too large.

b. if Anthony avoids $v$ such that $s(v) < K - 1$, this results in scenario 2. where Anthony avoids too few spiders. i.e. $K$ is too little.

Binary search for $K$.

For each $K$, use normal Dijkstra’s from $s$ to $t$ on the sub-graph, where vertices $v$ such that $s(v) < K$ are ignored. The failure condition is if Anthony does not make it in time.
Problem
Given a convex polygon $P$, create a smaller polygon $Q$ using a subset of points vertices from $P$ and maximize $\text{area}(Q) + \text{sum of values of vertices not in } Q$.

Statistics
0 solves / 14 attempted
Problem
Given a convex polygon $P$, maximize $M$

Solution
Let $f(i, j)$ denote the maximum possible score using only the vertices between $v[i]$ and $v[j]$ (inclusive)
Problem
Given a convex polygon P, maximize $M$

Solution
Suppose $v[k] \in Q$. Then maximum possible score is $f(i, k) + f(k, j) + \text{area}(v_i, v_k, v_j)$. 
Problem
Given a convex polygon $P$, maximize $M$

Solution
- Memoize recursion result
- Compute triangle area with cross product
- Time complexity: $O(n^3)$
Problem
Assign $n$ dogs to $m$ bowls while minimizing total waiting time.

Statistics
0 solves / 6 attempted

Solution
- Suppose all dogs finish eating at time $t$. Calculate the waiting time from assigning dog $i$ to bowl $j$. The minimum total waiting time can then be calculated using min-cost bipartite matching.
- Iterate through all possible end time.
Acknowledgement

Jury:

- Tony Cai
- Modan Han (Google)
- Zachary Friggstad (University of Alberta)
- Kent Williams-King (Brown University)
- Wen Li Looi (Google)
- Darko Aleksic (Assistant Coach, Microsoft)
Awesome job!

CPC has meetings every Wednesday (6pm to 8pm) and Saturday (10am to 3pm)

Next major contest: Calgary Collegiate Programming Contest (March 2019)