Solution Outlines

SWERC Judges

SWERC 2011

SWERC 2011 Statistics

First team solving the problem

Problem	1^{st} team solving	Time
D - Distributing Ballot Boxes	UPC-1	17
H - Peer Review	ENS Ulm 1	27
G - Non-negative Partial Sums	ETH Tautology Club	38
J - Remoteland	ETH Tautology Club	47
F - Guess the Numbers	ETH Tautology Club	55
I - Regular Convex Polygon	ENS Ulm 3	75
E - Game, Set and Match	ETH Tautology Club	143
C - Cybercrime Donut Investigation	UPC-1	213
A - Alphabet Soup	?	?
B - Coin Collecting	?	?

SWERC 2011 Statistics

Correct/Submissions Rate

Problem	Correct	Submissions	Rate
A - Alphabet Soup			
B - Coin Collecting			
C - Cybercrime Donut Investigation			
D - Distributing Ballot Boxes	24	98	24%
E - Game, Set and Match	12	17	71%
F - Guess the Numbers	19	38	50%
G - Non-negative Partial Sums	18	70	30%
H - Peer Review	30	127	24%
I - Regular Convex Polygon	9	64	14%
J - Remoteland	11	38	29%

SWERC 2011 Statistics

Correct/Submissions Rate

Problem	Correct	Submissions	Rate
A - Alphabet Soup	0+?	10	?%
B - Coin Collecting	0+?	5	?%
C - Cybercrime Donut Investigation	1+?	133	?%
D - Distributing Ballot Boxes	24	98	24%
E - Game, Set and Match	12	17	71%
F - Guess the Numbers	19	38	50%
G - Non-negative Partial Sums	18	70	30%
H - Peer Review	30	127	24%
I - Regular Convex Polygon	9	64	14%
J - Remoteland	11	38	29%

SWERC 2010 Statistics

We've been nice this time...

Problem	Correct	Submissions	Rate
A - Lawnmower	35	69	51%
B - Periodic points	0	6	0%
C - Comparing answers	6	122	5%
D - Fake scoreboard	1	26	4%
E - Palindromic DNA	1	5	20%
F - Jumping monkey	6	64	10%
G - Sensor network	0	10	0%
H - Assembly line	3	35	9%
I - Locks and keys	2	19	11%
J - 3-sided dice	3	106	3%

A - Alphabet Soup: solution is trivial ;)

Classification

- Difficulty: Hard
- Method: Combinatorics, Polya Counting

Task

1. Compute the smallest turning such that the rotation is valid.
2. Do the general counting with this observation.

A - Alphabet Soup: solution is trivial ;)

Classification

- Difficulty: Hard
- Method: Combinatorics, Polya Counting

Task

1. Compute the smallest turning such that the rotation is valid.
2. Do the general counting with this observation.

1. Sort the angles by increasing order ($P \log P$).
2. If «**no rotation**» return $S^P \mod 100,000,007$ ($\log P$ with fast exponentation).

A - Alphabet Soup: rotation

Approach 1:

- Take the P^2 distances between the points and try them all ($\Omega(P^3)$ TLE).

Approach 2:

- Fix a point.
- Consider the P distances to the rest ($\Omega(P^2)$ TLE).

Approach 3:

- Let the pasta pieces be situated at $p_i, i = 0, \ldots, P-1$.
- Consider the differences between adjacent pasta pieces $d_i = p_i - p_{i-1}$, where the indices are taken modulo P.
- Let s be the "string" formed by $d_1 d_2 \ldots d_P$.
- If s occurs twice in ss, then there are no possible rotations.
- If not, let k be the smallest positive number such that $d_{i+k} = d_i$ for all i.
- To compute this, use any of your favourite string-searching algorithm like Knuth-Morris-Pratt, Boyer-Moore or other $O(P)$ variants.
- Other possibilities: try all divisors of P or N, very fast in practice.

A - Alphabet Soup: eureka!

- Define the application g as the rotation by $d_1 + d_2 + \ldots + d_k$.
- g is a permutation of k cycles of $\frac{P}{k}$ elements each (k divides P).
- Use Burnside's Lemma and count how many fixed points there are by the action of g^i (g iterated i times):
  - For each of the cycles, g^i has $S^{\gcd(i, \frac{P}{k})}$ fixed points.
  - As we have k cycles, g^i has $S^{k \gcd(i, \frac{P}{k})}$ fixed points.
- The expected result is:

$$\frac{k}{P} \sum_{i=0}^{P/k-1} S^{k \gcd(i, \frac{P}{k})}$$

Compute $\frac{1}{P}$ applying the Fermat's Little Theorem to obtain $\frac{1}{P} = P^{100,000,007-2} \mod 100,000,007$.

B - Coin collecting

Statement

Pick the maximum number of envelopes subject to the following conditions:

- At most one envelope from each of m pairs.
- No non-empty subset contains an even number of coins from every country.

Categories: graphs, matroid intersection
Difficulty: very hard (I mean, impossible)

B - Coin collecting - Graph translation

- Construct an undirected graph $G = (V, E)$ with $V =$ countries.
- Think of envelopes as *edges* (each envelope has coins from two different countries).

B - Coin collecting - Graph translation

- Construct an undirected graph $G = (V, E)$ with $V =$ countries.
- Think of envelopes as *edges* (each envelope has coins from two different countries).
- We need to pick at most one edge from every pair.

B - Coin collecting - Graph translation

- Construct an undirected graph $G = (V, E)$ with $V =$ countries.
- Think of envelopes as *edges* (each envelope has coins from two different countries).
- We need to pick at most one edge from every pair.
- No non-empty subset of E has all vertices of even degree.

What to make of the last condition?

B - Coin collecting - Graph translation

- Construct an undirected graph $G = (V, E)$ with $V =$ countries.
- Think of envelopes as *edges* (each envelope has coins from two different countries).
- We need to pick at most one edge from every pair.
- No non-empty subset of E has all vertices of even degree.

What to make of the last condition?
All vertices in a subset of E have even degree $\Leftrightarrow$ we can find a circuit in E!
The condition becomes: E is **acyclic**, i.e E is a forest (union of vertex-disjoint trees).

B - Coin collecting - First attempt

Problem, restated

Pick as large a subset E of edges as possible subject to the following conditions:

- At most one edge from each of m pairs.
- E is acyclic.

B - Coin collecting - First attempt

Problem, restated

Pick as large a subset E of edges as possible subject to the following conditions:

- At most one edge from each of m pairs.
- E is acyclic.

Doesn't even look like this has a poly-time solution!

B - Coin collecting - First attempt

Problem, restated

Pick as large a subset E of edges as possible subject to the following conditions:

- At most one edge from each of m pairs.
- E is acyclic.

Doesn't even look like this has a poly-time solution!
Hint: keep making choices as new pairs of edges arrive, and "change your mind" about the previous choices if necessary

B - Coin collecting - First attempt

Problem, restated

Pick as large a subset E of edges as possible subject to the following conditions:

- At most one edge from each of m pairs.
- E is acyclic.

Doesn't even look like this has a poly-time solution!
Hint: keep making choices as new pairs of edges arrive, and "change your mind" about the previous choices if necessary
Algorithm:

1. $E \leftarrow \emptyset$
2. **for each** pair of candidate edges (e_1, e_2)
3. **if** $E + e_1$ is acyclic, $E \leftarrow E + e_1$;
4. **else if** $E + e_2$ is acyclic, $E \leftarrow E + e_2$;
5. **else** change your mind?

B - Coin collecting - Questioning previous choices

We try to add edge e to E:

- if $E + e$ is acyclic, done
- otherwise pick an edge f_1 in the *unique* cycle $C(E + e)$ of $E + e$ and remove it to form $E + e - f_1$.
- After removing f_1, the "companion" edge f_1' becomes free.
- If $E + e - f_1 + f_1'$ acyclic, done (E has been increased by one).
- Otherwise pick f_2 in a cycle of $E + e - f_1$ and repeat...

B - Coin collecting - Questioning previous choices

We try to add edge e to E:

- if $E + e$ is acyclic, done
- otherwise pick an edge f_1 in the *unique* cycle $C(E + e)$ of $E + e$ and remove it to form $E + e - f_1$.
- After removing f_1, the "companion" edge f_1' becomes free.
- If $E + e - f_1 + f_1'$ acyclic, done (E has been increased by one).
- Otherwise pick f_2 in a cycle of $E + e - f_1$ and repeat...

This has a similar flavor to the bipartite matching algorithms.

B - Coin collecting - Almost there

New goal

Find a sequence of edges $e, f_1, \ldots, f_t$ such that $(E + e) - f_1 + f_1' - f_2 + f_2' \ldots - f_t + f_t'$ is acyclic and has size $|E| + 1$.
(this is called an *augmenting path*)

To do so, we want all graphs $(E + e) - f_1, (E + e) - f_1 + f_1' - f_2, \ldots$ to be acyclic.

B - Coin collecting - Almost there

New goal

Find a sequence of edges $e, f_1, \ldots, f_t$ such that $(E + e) - f_1 + f_1' - f_2 + f_2' \ldots - f_t + f_t'$ is acyclic and has size $|E| + 1$.
(this is called an *augmenting path*)

To do so, we want all graphs $(E + e) - f_1, (E + e) - f_1 + f_1' - f_2, \ldots$ to be acyclic.

Caveat

- The graph changes as we add/remove edges.
- We do **not** want to backtrack over the new graphs formed, so we'd like to check for cycles in $E + f_i'$ alone.
- But it is possible that the unique cycle $C(E + f_i')$ involves a previously removed $f_j, j < i$. Cannot remove an edge twice!

How to solve?

B - Coin collecting - Almost there

New goal

Find a sequence of edges $e, f_1, \ldots, f_t$ such that $(E + e) - f_1 + f_1' - f_2 + f_2' \ldots - f_t + f_t'$ is acyclic and has size $|E| + 1$.
(this is called an *augmenting path*)

To do so, we want all graphs $(E + e) - f_1, (E + e) - f_1 + f_1' - f_2, \ldots$ to be acyclic.

Caveat

- The graph changes as we add/remove edges.
- We do **not** want to backtrack over the new graphs formed, so we'd like to check for cycles in $E + f_i'$ alone.
- But it is possible that the unique cycle $C(E + f_i')$ involves a previously removed $f_j, j < i$. Cannot remove an edge twice!

How to solve? Just enforce the last condition: we only add edges whose cycles do not involve previously removed edges.

B - Coin collecting - Solution: good old BFS

- Build a *directed* graph $G'(E)$ with vertices labeled $+e$ or $-f$ (for $e, f \in E$)

B - Coin collecting - Solution: good old BFS

- Build a *directed* graph $G'(E)$ with vertices labeled $+e$ or $-f$ (for $e, f \in E$)
- Put an edge from $+e$ to $-f$ for all $f \in C(E + e)$.
- Put an edge from $-f$ to $+f'$ (the companion edge).

B - Coin collecting - Solution: good old BFS

- Build a *directed* graph $G'(E)$ with vertices labeled $+e$ or $-f$ (for $e, f \in E$)
- Put an edge from $+e$ to $-f$ for all $f \in C(E + e)$.
- Put an edge from $-f$ to $+f'$ (the companion edge).
- Sources: the two new edges.
- Sinks: all edges with out-degree 0.
- Look for a path from a source to a sink.

B - Coin collecting - Solution: good old BFS

- Build a *directed* graph $G'(E)$ with vertices labeled $+e$ or $-f$ (for $e, f \in E$)
- Put an edge from $+e$ to $-f$ for all $f \in C(E + e)$.
- Put an edge from $-f$ to $+f'$ (the companion edge).
- Sources: the two new edges.
- Sinks: all edges with out-degree 0.
- Look for a path from a source to a sink.

If in our path, $C(E + f_i')$ involved a previously removed f_j, there would be a *shortcut*. But if the path is **shortest**, there can be no shortcuts!

B - Coin collecting - Solution: good old BFS

- Build a *directed* graph $G'(E)$ with vertices labeled $+e$ or $-f$ (for $e, f \in E$)
- Put an edge from $+e$ to $-f$ for all $f \in C(E + e)$.
- Put an edge from $-f$ to $+f'$ (the companion edge).
- Sources: the two new edges.
- Sinks: all edges with out-degree 0.
- Look for a path from a source to a sink.

If in our path, $C(E + f_i')$ involved a previously removed f_j, there would be a *shortcut*. But if the path is **shortest**, there can be no shortcuts! DFS **won't** work.

B - Coin collecting - Final remarks

- It can be proven that if E is not optimal, there is an augmenting path.
- The algorithm just described can be easily implemented in $O(n^3)$.
- Time limits not tight.
- The algorithm resembles matching algorithms, and in fact both are a special case of Edmond's *matroid intersection* (or *matroid partitioning*) algorithm.

C - Donuts Scene Investigation

Underlying problem

- n fixed points in the plane $(p^1, \ldots, p^n)$.
- q query points.
- For each of the q query points, return the closest (in Manhattan distance) of the n fixed points.

Categories: Data structures, geometry
Difficulty: Hard

C - Donuts Scene Investigation - First observations

We preprocess the n fixed points to create a data structure that can answer quickly each of the queries.

C - Donuts Scene Investigation - First observations

We preprocess the n fixed points to create a data structure that can answer quickly each of the queries.
We will first assume that all of the query points are at the right of all the n fixed points:

- Let the query point be (x, y). The answer will be

$$\min_i \left(x - p_x^i + \left| y - p_y^i \right| \right)$$

- This can be written as

$$\min \left(\min_{y \geq p_y^i} \left(x - p_x^i + y - p_y^i \right), \min_{y \leq p_y^i} \left(x - p_x^i + p_y^i - y \right) \right)$$

C - Donuts Scene Investigation - First observations

We preprocess the n fixed points to create a data structure that can answer quickly each of the queries.
We will first assume that all of the query points are at the right of all the n fixed points:

- Let the query point be (x, y). The answer will be

$$\min_i \left(x - p_x^i + \left| y - p_y^i \right| \right)$$

- This can be written as

$$\min \left(\min_{y \geq p_y^i} \left(x - p_x^i + y - p_y^i \right), \min_{y \leq p_y^i} \left(x - p_x^i + p_y^i - y \right) \right)$$

- It suffices then to compute the point above (x, y) minimizing $-p_x^i + p_y^i$, and the point below (x, y) minimizing $-p_x^i - p_y^i$

C - Donuts Scene Investigation - Solution for simpler case

This suggests how to create a data structure that solves this simpler case:

- Sort the p^i by the y coordinate.

C - Donuts Scene Investigation - Solution for simpler case

This suggests how to create a data structure that solves this simpler case:

- Sort the p^i by the y coordinate.
- Keep a table that for each of the p^i holds the minimum of $-p_x^i + p_y^i$ for points above it, and a another one that holds the minimum of $-p_x^i - p_y^i$ for points below it. These tables can be computed with two linear passes over the input once it is sorted by the y coordinate.

C - Donuts Scene Investigation - Solution for simpler case

This suggests how to create a data structure that solves this simpler case:

- Sort the p^i by the y coordinate.
- Keep a table that for each of the p^i holds the minimum of $-p^i_x + p^i_y$ for points above it, and a another one that holds the minimum of $-p^i_x - p^i_y$ for points below it. These tables can be computed with two linear passes over the input once it is sorted by the y coordinate.
- To answer a query, perform a binary search to find the first of the p^i above (x, y), and the first of the p^i below (x, y), and use the appropriate entries of the tables to compute the answer.

C - Donuts Scene Investigation - Solution for simpler case

This suggests how to create a data structure that solves this simpler case:

- Sort the p^i by the y coordinate.
- Keep a table that for each of the p^i holds the minimum of $-p^i_x + p^i_y$ for points above it, and a another one that holds the minimum of $-p^i_x - p^i_y$ for points below it. These tables can be computed with two linear passes over the input once it is sorted by the y coordinate.
- To answer a query, perform a binary search to find the first of the p^i above (x, y), and the first of the p^i below (x, y), and use the appropriate entries of the tables to compute the answer.

This can also be used in the case in which the query point is guaranteed to be at the left of all the p^i, but using instead the values of $p^i_x + p^i_y$ and $p^i_x - p^i_y$ to create our tables.

C - Donuts Scene Investigation - General solution

To solve the general problem, we decompose each query into a series of queries falling in the simpler case:

- Create a range tree for the p^i according to the x coordinate.

C - Donuts Scene Investigation - General solution

To solve the general problem, we decompose each query into a series of queries falling in the simpler case:

- Create a range tree for the p^i according to the x coordinate.
- For each of the nodes of the tree, sort the corresponding points by their y coordinate, and compute the tables described in the solution for the simpler case.

C - Donuts Scene Investigation - General solution

To solve the general problem, we decompose each query into a series of queries falling in the simpler case:

- Create a range tree for the p^i according to the x coordinate.
- For each of the nodes of the tree, sort the corresponding points by their y coordinate, and compute the tables described in the solution for the simpler case.
- To answer a query, use the range tree to divide the p^i into $O(\log n)$ groups, each of them completely at the left of (x, y), or completely at the right of (x, y).

C - Donuts Scene Investigation - General solution

To solve the general problem, we decompose each query into a series of queries falling in the simpler case:

- Create a range tree for the p^i according to the x coordinate.
- For each of the nodes of the tree, sort the corresponding points by their y coordinate, and compute the tables described in the solution for the simpler case.
- To answer a query, use the range tree to divide the p^i into $O(\log n)$ groups, each of them completely at the left of (x, y), or completely at the right of (x, y).
- For each of the groups, use the tables for the corresponding node, and our solution for the simpler case, to determine the closest point in Manhattan distance.

C - Donuts Scene Investigation - General solution

To solve the general problem, we decompose each query into a series of queries falling in the simpler case:

- Create a range tree for the p^i according to the x coordinate.
- For each of the nodes of the tree, sort the corresponding points by their y coordinate, and compute the tables described in the solution for the simpler case.
- To answer a query, use the range tree to divide the p^i into $O(\log n)$ groups, each of them completely at the left of (x, y), or completely at the right of (x, y).
- For each of the groups, use the tables for the corresponding node, and our solution for the simpler case, to determine the closest point in Manhattan distance.
- Take the best answer over all the groups.

We can also look at this as an augmented 2D range tree.

C - Donuts Scene Investigation - Complexity

- Building the range tree for the x coordinate takes $O(n \log n)$ time.

C - Donuts Scene Investigation - Complexity

- Building the range tree for the x coordinate takes $O(n \log n)$ time.
- As each point appears in $O(\log n)$ nodes, sorting the points in each node by y coordinate takes $O(n \log n^2)$ time(or $O(n \log n)$ using a linear-time merge algorithm).

C - Donuts Scene Investigation - Complexity

- Building the range tree for the x coordinate takes $O(n \log n)$ time.
- As each point appears in $O(\log n)$ nodes, sorting the points in each node by y coordinate takes $O(n \log n^2)$ time(or $O(n \log n)$ using a linear-time merge algorithm).
- Computing the tables for each of the nodes takes time linear in the number of points in the node. As each point appears in $O(\log n)$ nodes, the total time for computing the tables is $O(n \log n)$.

C - Donuts Scene Investigation - Complexity

- Building the range tree for the x coordinate takes $O(n \log n)$ time.
- As each point appears in $O(\log n)$ nodes, sorting the points in each node by y coordinate takes $O(n \log n^2)$ time(or $O(n \log n)$ using a linear-time merge algorithm).
- Computing the tables for each of the nodes takes time linear in the number of points in the node. As each point appears in $O(\log n)$ nodes, the total time for computing the tables is $O(n \log n)$.
- Answering a query involves solving $O(\log n)$ problems for the simpler case, each of which takes the cost of a binary search, $O(\log n)$.

C - Donuts Scene Investigation - Complexity

- Building the range tree for the x coordinate takes $O(n \log n)$ time.
- As each point appears in $O(\log n)$ nodes, sorting the points in each node by y coordinate takes $O(n \log n^2)$ time(or $O(n \log n)$ using a linear-time merge algorithm).
- Computing the tables for each of the nodes takes time linear in the number of points in the node. As each point appears in $O(\log n)$ nodes, the total time for computing the tables is $O(n \log n)$.
- Answering a query involves solving $O(\log n)$ problems for the simpler case, each of which takes the cost of a binary search, $O(\log n)$.
- Total cost: $O(n \log n + q \log n^2)$

C - Donuts Scene Investigation - Ideas for alternative efficient approaches

- Rotate all of the input by 45 degrees. Put the n given points in a 2D range tree. This is a binary tree sorted by the x coordinate: each leave corresponds to a point and each internal node to an interval of consecutive points. At each internal node, store the list of points in the interval, this time sorted by their y coordinate. This allows to determine if a rectangle has non-empty intersection with the points in $O((\log n)^2)$ time. Then, for each of the q queries, use binary search to find the smallest non-empty square with its centre on the query point. Half of its side length will be the answer.

C - Donuts Scene Investigation - Ideas for alternative efficient approaches

- Rotate all of the input by 45 degrees. Put the n given points in a 2D range tree. This is a binary tree sorted by the x coordinate: each leave corresponds to a point and each internal node to an interval of consecutive points. At each internal node, store the list of points in the interval, this time sorted by their y coordinate. This allows to determine if a rectangle has non-empty intersection with the points in $O((\log n)^2)$ time. Then, for each of the q queries, use binary search to find the smallest non-empty square with its centre on the query point. Half of its side length will be the answer.
- Build a Voronoi diagram for the Manhattan distance (much harder).

C - Donuts Scene Investigation - Other notes on complexity

- It is possible to improve on the described solution by linking the tables for different nodes of the tree so that we only have to perform a binary search in one of them, and successive solutions of the subproblems take only constant time. This is known in the data structures literature as "fractional cascading", and it brings the time cost down to $O(n \log n + q \log n)$

C - Donuts Scene Investigation - Other notes on complexity

- It is possible to improve on the described solution by linking the tables for different nodes of the tree so that we only have to perform a binary search in one of them, and successive solutions of the subproblems take only constant time. This is known in the data structures literature as "fractional cascading", and it brings the time cost down to $O(n \log n + q \log n)$
- It is possible to build cases (e.g. a diamond, with the query point in the centre, and the n points at the border) where a pruning-based solution will have to inspect almost all points, and therefore won't be efficient.

D - Distributing Ballot Boxes

Statement

Distribute boxes among cities:

- At least one in every city.
- Minimize the box most people are assigned to.

Difficulty: easy/medium.

Remark: At every city people must be distributed proportionally in an optimal assignment.

D - Distributing Ballot Boxes

First approach - Give to the most needed

- Give 1 box to each city.
- For each of the remaining boxes:
  1. Find the city which needs it the most ($O(N) = TLE$, $O(log(N)) = AC$ with a priority queue).
  2. Give the box to that city.

Complexity: $O(N + B \log N)$

D - Distributing Ballot Boxes

Second approach - Binary search for the answer

- Take $M =$ maximum number of people assigned to vote in one box.
- How many ballot boxes will we need? Computation in $O(N)$.
- If less than M, try larger M.
- If more than M, try smaller M.

Using **Binary search** we get the solution in $O(Nlog(B))$.

E - Game, Set and Match

Statement

Compute the probability of winning a given game, a given set and a given match if each point is won with probability p.

Categories: probability, dynamic programming
Difficulty: medium

E - Game, Set and Match - Rule simplification

The rules are actually very simple:

- To win a game, you need to win ≥ 4 points by a margin of 2.
- To win a set, you need to win ≥ 6 games by a margin of 2, except if the score reaches $6-6$ (tiebreak).
- To win the tiebreak, you need to win ≥ 7 points by a margin of 2.
- To win the match, you need to win ≥ 2 sets.

Common subproblem

Compute $a(p_1)$, the probability of being the first to win two consecutive "points", when tied (if each "point" is won with probability p_1). Compute $b(n, p_1, p_2)$, the probability of winning n "points" by a margin of 2 if:

- each "point" is won with probability p_1
- when the score is $n-n$, the probability of eventualy winning is p_2.

E - Game, Set and Match - What needs to be done

Common subproblem

Compute $a(p_1)$, the probability of being the first to win two consecutive "points", when tied (if each "point" is won with probability p_1). Compute $b(n, p_1, p_2)$, the probability of winning n "points" by a margin of 2 if:

- each "point" is won with probability p_1
- when the score is $n - n$, the probability of eventualy winning is p_2.

Then

- Pr[win game] $= b(4, p, a(p))$.
- Pr[win tiebreak] $= b(7, p, a(p))$.
- Pr[win set] $= b(6, \text{Pr[win game]}, \text{Pr[win tiebreak]})$.
- Pr[win match] $= r^2 + 2r(1 - r)$, where $r = \text{Pr[win set]}$.

E - Game, Set and Match - Winning probability at deuce

Let $q = a(p)$.

- Recall that each point is won with probability p.
- With probability p^2, we win directly.
- With probability $2p(1-p)$, we go back to deuce.
- So q must satisfy $q = p^2 + p(1-p)q$, i.e.

$$q = \frac{p^2}{1 - 2p(1-p)}.$$

A sufficiently accurate simulation would also work here.

E - Game, Set and Match - Computing $b(n, p_1, p_2)$ by DP

Let $f(i, j) =$ probability of winning at score $i - j$. Base cases:

- $f(i, j) = 1$ if $i \geq n$ and $i \geq j + 2$.
- $f(i, j) = 0$ if $j \geq n$ and $j \geq i + 2$.
- $f(n, n) = p_2$.
- $f(n, n-1) = p_1 + (1 - p_1) \cdot f(n, n)$.
- $f(n-1, n) = p_1 \cdot f(n, n)$.

Remaining cases:

$$f(i, j) = p_1 \cdot f(i+1, j) + (1 - p_1) \cdot f(i, j+1).$$

Fill a table $f[][]$ in a top-down fashion using these equations.
The answer is $f(0, 0)$.

Another approach (ENS Ulm 3)

```
pj = pow(p,4) + 4*pow(p,4)*(1-p) + 10*pow(p,4)*
  pow(1-p,2) + 20*pow(p,5)* pow(1-p,3)/(2*p*p-2*p+1);

pt = pow(p,7) + 7*pow(p,7)*pow(1-p,1) +
  28*pow(p,7)*pow(1-p,2) + 84*pow(p,7)*pow(1-p,3) +
  210*pow(p,7)*pow(1-p,4) +462*pow(p,7)*pow(1-p,5) +
  924*pow(p,8)*pow(1-p,6)/(2*p*p-2*p+1);

ps = pow(pj,6) + 6*pow(pj,6)*pow(1-pj,1) +
  21*pow(pj,6)*pow(1-pj,2) +
  56*pow(pj,6)*pow(1-pj,3) +
  126*pow(pj,6)*pow(1-pj,4) +
  252*pow(pj,5)*pow(1-pj,5)*(pj*pj+2*pj*(1-pj)*pt);

pm = ps*ps + 2*ps*ps*(1-ps);
```

F - Guess the Numbers

Classification

- Difficulty: Easy
- Method: Backtracking

Task

Given an arithmetic expression e with unknowns $x_1 \ldots x_n$ and some values $v_1 \ldots v_n$, compute if there is an assignment of the values to the unknowns so that the expression evaluates to a specific result.

F - Guess the Numbers

Steps

1. **Parse the expression.** Very simple syntax:

$$\begin{aligned} \mathbf{e} ::= \quad & \text{“}\textit{unknown}\text{”} \text{ (lowercase letter)} \\ & \mid (\mathbf{e_1} + \mathbf{e_2}) \\ & \mid (\mathbf{e_1} - \mathbf{e_2}) \\ & \mid (\mathbf{e_1} * \mathbf{e_2}) \end{aligned}$$

2. **Try all the possible assignments.**
   n values for n unknowns $\Rightarrow$ $n!$ posibilites, corresponding to the $n!$ permutations of the values.

F - Guess the Numbers

Solution

```
e = Parse the expression
For each possible permutation v of the values
  r = Evaluate(e,v)
  If r == result then print "YES"
If result not found then print "NO"
```

Cost $O(n \cdot n!)$ *(this problem is tractable because* $1 \leq n \leq 5$ *is* **very** *small)*

G - Non-negative Partial Sums

Statement:

- You are given a sequence of n integers.
- How many cyclic shifts of the sequence have the property that all partial sums are non-negative?

Categories: greedy *Difficulty:* medium

G - Non-negative Partial Sums - Idea

- Precalculate arrays which allow to evaluate in constant time if some cyclic shift produces only non-negative partial sums.
- We define a start position as the position which becomes the first position in the array after a cyclic shift.
- Define $A[i] = \sum_{j=1}^{i} a[j]$ and $B[i] = \sum_{j=i}^{n} a[j]$
- Define $C[i] = \min(A[1], A[2], \ldots, A[i])$.
- Define $D[i] = \min(a[i], a[i] + a[i+1], \ldots, a[i] + a[i+1] + \ldots + a[n])$.
- Position i is a valid start position if $D[i] \geq 0$ and $B[i] + C[i-1] \geq 0$ (if $i = 1$ we only need to check if $D[1] \geq 0$).
- Loop over all n positions and check this condition.

G - Non-negative Partial Sums - Details

- Arrays A, B, C and D can be calculated in $O(n)$ using an equivalent recursive definition.
- Let $A[0] = 0$. Then $A[i] = A[i-1] + a[i]$ for $0 < i \leq n$.
- Let $B[n+1] = 0$. Then $B[i] = B[i+1] + a[i]$ for $1 \leq i \leq n$.
- Let $C[0] = \infty$. Then $C[i] = \min(A[i], C[i-1])$ for $0 < i \leq n$.
- Let $D[n+1] = \infty$. Then $D[i] = \min(a[i], a[i] + D[i+1])$.

H - Peer Review

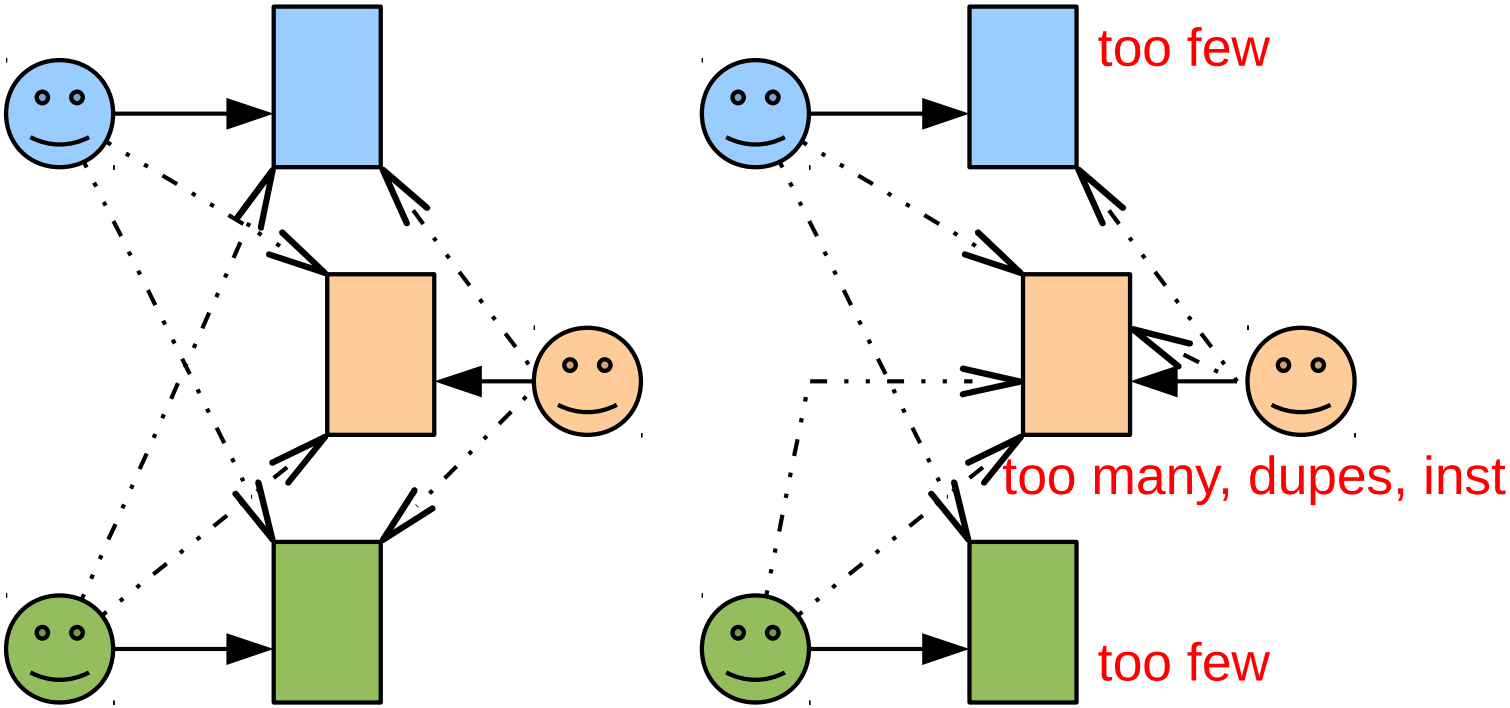
Statement

- Each paper should be reviewed by K scientists.
- Scientists should not review papers written by people they collaborate with (including themselves), or review the same paper more than once.
- If a paper is being reviewed too much, too little, or by the wrong people - tell the organizers.

Categories: graphs
Difficulty: easy

H - Peer Review

Sample input (illustrated):



Output for sample input:

- `NO PROBLEMS FOUND` (left)
- `3 PROBLEMS FOUND` (right)

H - Peer Review - Idea

Go from author-centric (author $\rightarrow$ list of reviews)
to paper-centric (paper $\rightarrow$ list of reviewers) ...
... and check for rule violations.

H - Peer Review - Idea

Go from author-centric (author $\rightarrow$ list of reviews)
to paper-centric (paper $\rightarrow$ list of reviewers) ...
... and check for rule violations.

The ith paper is *not* being reviewed correctly if it is being...

1. reviewed too much or too litle $\equiv$ number of reviewers for $i \neq K$
2. reviewed by the wrong people: if $r_1 \ldots r_k$ are reviewers for i,
   1. same reviewer twice $\equiv \exists\ u \neq v$ with $r_u = r_v$
   2. author or colleague as reviewer $\equiv \exists\ j$ with $inst(r_j) = inst(i)$

H - Peer Review - Rules to check

If `revs` is a vector with all the reviewers for paper `p`, and `inst` contains the institutions for each author,

```
bool good(int p, vector<int> revs, vector<int> inst) {
    // check rule 1
    if (revs.size() != k) return false;

    // check rules 2.1 and 2.2
    sort(revs, revs.begin(), revs.end());
    for (int i=(int)revs.size()-1; i>=0; i--) {
        if (i>0 && revs[i] == revs[i-1]) return false;
        if (inst[revs[i]] == inst[p]) return false;
    }
    return true;
}
```

H - Peer Review - Complexity

- The input itself is $O(N \cdot K)$, and it can be read in linear time, since a hash-map allows institution names to be looked up in $O(1)$.
- Processing of each paper can be done in time $O(K \cdot log(K))$ - due to sorting. Without sorting, $O(K^2)$ time would be required to check for duplicate reviewers.
- K was limited to small values, so both strategies were accepted by the judge.

H - Peer Review - Trivia

A very simple algorithm to assign good reviews, if there are many small institutions and authors from different institutions do not collaborate:

- shuffle all authors individually
- shuffle all institutions (keeping author order)
- let j be the number of authors in the largest institution
- author i is asked to review papers $i + (j \cdot 1) \ldots i + (j \cdot k)$ **mod** N

I am currently using it to assign peer reviews for my students!

I - Regular Convex Polygon

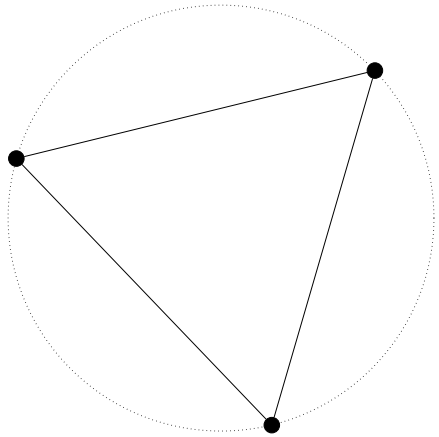
Statement

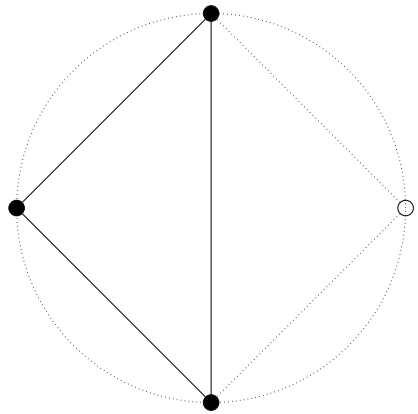
- Find a regular convex polygon with as few vertices as possible that has the given three points as vertices
- It is assured that there exists always a solution involving a regular convex polygon with at most 1000 vertices

Categories: geometry
Difficulty: medium

I - Regular Convex Polygon

Sample input (illustrated):





I - Regular Convex Polygon - Idea

Calculate the circumcircle of the three given points. This is also the circumcircle of all points of the regular convex polygon we look for!

I - Regular Convex Polygon - Idea

Calculate the circumcircle of the three given points. This is also the circumcircle of all points of the regular convex polygon we look for!

The vertices of the regular convex polygon should evenly divide the circle into n segments.

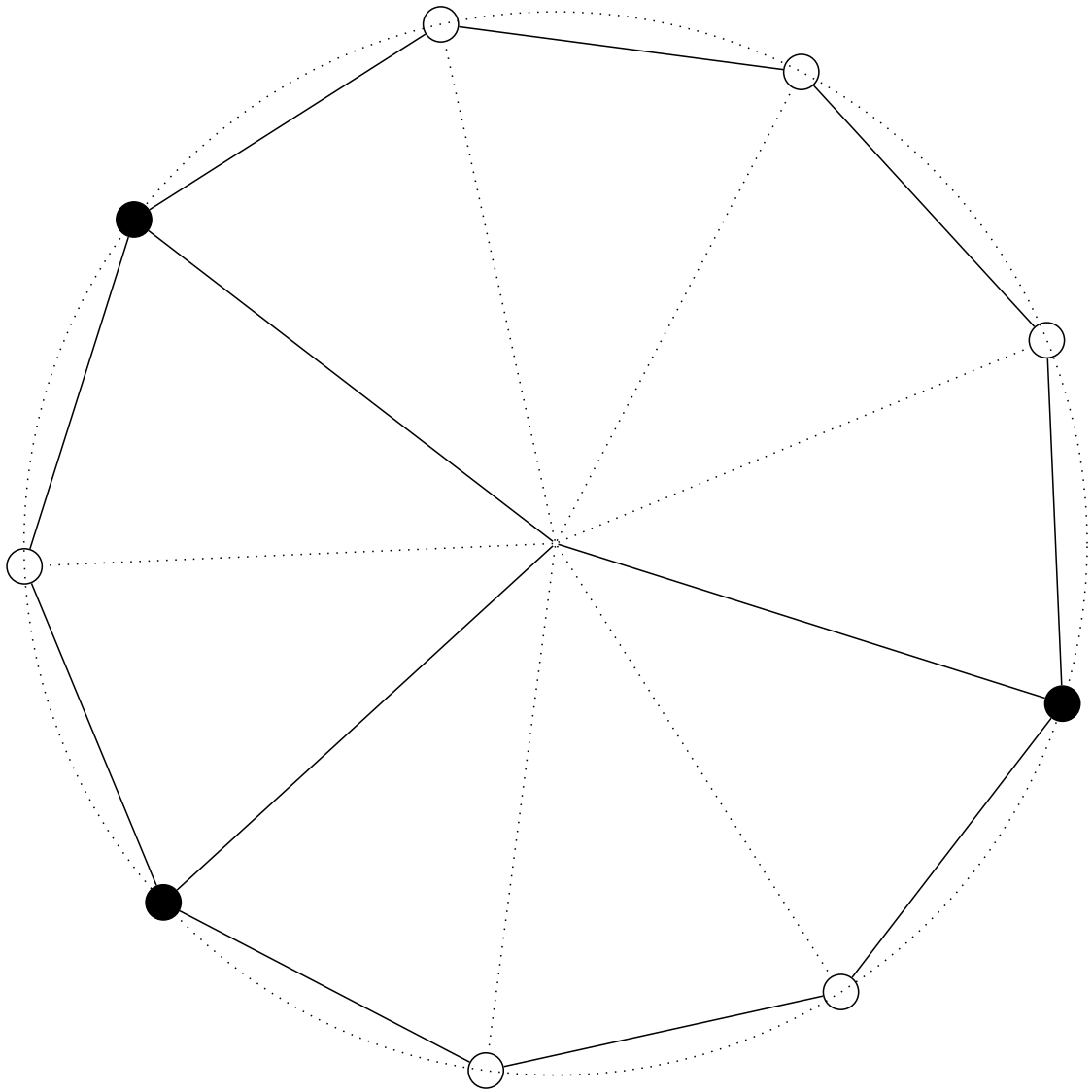
I - Regular Convex Polygon - Idea

Calculate the circumcircle of the three given points. This is also the circumcircle of all points of the regular convex polygon we look for!

The vertices of the regular convex polygon should evenly divide the circle into n segments.

Brute force over the number of points the polygon has. For a certain number of points n, check if the given three points divide the circle into segments of a size which is a multiple of the segment size for n points.

I - Regular Convex Polygon - Example



I - Regular Convex Polygon - Even simpler

Compute the angles of the triangle, α,β,γ.

Exercise: They should be integer multiples of $\frac{\pi}{n}$.

J - Remoteland

Statement

What is the largest possible divisor of $n!$ which is a square?

Categories: greedy, number theory
Difficulty: easy, medium

J - Remoteland

- Precalculate all primes which are less than the maximum allowed n using Erathostenes' Sieve.
- For all primes p_i less or equal than n, compute the exponent α_i of p_i in the prime decomposition of $n!$ using the formula:

$$\alpha_i = \left\lfloor \frac{n}{p_i} \right\rfloor + \left\lfloor \frac{n}{p_i^2} \right\rfloor + \left\lfloor \frac{n}{p_i^3} \right\rfloor + \dots$$

- If the parity is even, we do nothing. The exponent of p_i should be α_i.
- If the parity is odd, the exponent should be $\alpha_i - 1$ (equivalent as not taking p_i from the set).
- All exponentiations calculated using fast (logarithmic) exponentiation.
- Careful about fast I/O!

The full SWERC 2011 statistics

Correct/Submissions Rate

Problem	Correct	Submissions	Rate
A - Alphabet Soup	0	0	-
B - Coin Collecting	0	0	-
C - Cybercrime Donut Investigation	2	133	1.5%
D - Distributing Ballot Boxes	24	98	24%
E - Game, Set and Match	12	17	71%
F - Guess the Numbers	19	38	50%
G - Non-negative Partial Sums	18	70	30%
H - Peer Review	30	127	24%
I - Regular Convex Polygon	9	64	14%
J - Remoteland	11	38	29%