NCPC 2022
Presentation of solutions

2022-10-08

Problems prepared by

- Per Austrin (KTH Royal Institute of Technology)
- Atli Fannar Franklín (University of Iceland)
- Nils Gustafsson (KTH Royal Institute of Technology)
- Johan Sannemo (Stripe)
- Bergur Snorrason (University of Iceland)

Test solvers

- Joakim Blikstad
- Simon Lindholm
- Björn Martinsson

C — Coffee Cup Combo

Problem

You are given a sequence of lectures, some of which have coffee machines. You want to drink coffee at as many lectures as possible. You can bring coffee from machines, but you can only hold at most two cups at a time.

Statistics at 4-hour mark: 283 submissions, 198 accepted, first after 00:03

C — Coffee Cup Combo

Problem

You are given a sequence of lectures, some of which have coffee machines. You want to drink coffee at as many lectures as possible. You can bring coffee from machines, but you can only hold at most two cups at a time.

Solution

1. Iterate from left to right, and keep a variable `coffee` that stores how many cups you are holding.

Statistics at 4-hour mark: 283 submissions, 198 accepted, first after 00:03

C — Coffee Cup Combo

Problem

You are given a sequence of lectures, some of which have coffee machines. You want to drink coffee at as many lectures as possible. You can bring coffee from machines, but you can only hold at most two cups at a time.

Solution

1. Iterate from left to right, and keep a variable `coffee` that stores how many cups you are holding.
2. When encountering a coffee machine: `answer += 1` and `coffee = 2`.

Statistics at 4-hour mark: 283 submissions, 198 accepted, first after 00:03

C — Coffee Cup Combo

Problem

You are given a sequence of lectures, some of which have coffee machines. You want to drink coffee at as many lectures as possible. You can bring coffee from machines, but you can only hold at most two cups at a time.

Solution

1. Iterate from left to right, and keep a variable `coffee` that stores how many cups you are holding.
2. When encountering a coffee machine: `answer += 1` and `coffee = 2`.
3. When entering a no-coffee room, do nothing if `coffee == 0`. Otherwise, `coffee -= 1` and `answer += 1`.

Statistics at 4-hour mark: 283 submissions, 198 accepted, first after 00:03

D — Disc District

Problem

Given a radius r, find a point with integer coordinates that is as close as possible to the origin, while being further than distance r away.

Statistics at 4-hour mark: 278 submissions, 169 accepted, first after 00:04

D — Disc District

Problem

Given a radius r, find a point with integer coordinates that is as close as possible to the origin, while being further than distance r away.

Solution

1. Method 1: try all x-coordinates from 0 to r, and use binary search or math to find the best y-coordinate.

Statistics at 4-hour mark: 278 submissions, 169 accepted, first after 00:04

D — Disc District

Problem

Given a radius r, find a point with integer coordinates that is as close as possible to the origin, while being further than distance r away.

Solution

1. Method 1: try all x-coordinates from 0 to r, and use binary search or math to find the best y-coordinate.
2. Method 2: $x^2 + y^2 > r^2 \iff x^2 + y^2 \geq r^2 + 1$. Choosing $(x, y) = (r, 1)$ will give the optimal distance.

Statistics at 4-hour mark: 278 submissions, 169 accepted, first after 00:04

H — Highest Hill

Problem

Given a sequence of integers $h_1, h_2, \cdots, h_n$, a *hill* is defined as an interval where $h_i \leq h_{i+1} \leq \cdots \leq h_j \geq h_{j+1} \geq \cdots \geq h_k$. The height of a hill is defined as $\min(h_j - h_i, h_j - h_k)$. Find the highest hill.

Solution

Statistics at 4-hour mark: 456 submissions, 121 accepted, first after 00:08

H — Highest Hill

Problem

Given a sequence of integers $h_1, h_2, \cdots, h_n$, a *hill* is defined as an interval where $h_i \leq h_{i+1} \leq \cdots \leq h_j \geq h_{j+1} \geq \cdots \geq h_k$. The height of a hill is defined as $\min(h_j - h_i, h_j - h_k)$. Find the highest hill.

Solution

1. For every index i, calculate $L(i)$, the minimum j such that $h_j \leq h_{j+1} \leq \cdots \leq h_i$.

Statistics at 4-hour mark: 456 submissions, 121 accepted, first after 00:08

H — Highest Hill

Problem

Given a sequence of integers $h_1, h_2, \cdots, h_n$, a *hill* is defined as an interval where $h_i \leq h_{i+1} \leq \cdots \leq h_j \geq h_{j+1} \geq \cdots \geq h_k$. The height of a hill is defined as $\min(h_j - h_i, h_j - h_k)$. Find the highest hill.

Solution

1. For every index i, calculate $L(i)$, the minimum j such that $h_j \leq h_{j+1} \leq \cdots \leq h_i$.
2. Similarly, calculate $R(i)$, the maximum j such that $h_i \geq \cdots \geq h_j$.

Statistics at 4-hour mark: 456 submissions, 121 accepted, first after 00:08

H — Highest Hill

Problem

Given a sequence of integers $h_1, h_2, \cdots, h_n$, a *hill* is defined as an interval where $h_i \le h_{i+1} \le \cdots \le h_j \ge h_{j+1} \ge \cdots \ge h_k$. The height of a hill is defined as $\min(h_j - h_i, h_j - h_k)$. Find the highest hill.

Solution

1. For every index i, calculate $L(i)$, the minimum j such that $h_j \le h_{j+1} \le \cdots \le h_i$.
2. Similarly, calculate $R(i)$, the maximum j such that $h_i \ge \cdots \ge h_j$.
3. The height of the hill with peak at i is $\min(h_i - h_{L(i)}, h_i - h_{R(i)})$. Take the maximum of these.

Statistics at 4-hour mark: 456 submissions, 121 accepted, first after 00:08

A — Ace Arbiter

Problem

Check that a sequence of scores could have a appeared as a subsequence in a ping pong game.

Statistics at 4-hour mark: 608 submissions, 111 accepted, first after 00:11

A — Ace Arbiter

Problem

Check that a sequence of scores could have a appeared as a subsequence in a ping pong game.

Solution

1. We want to know who has what score. If Bob is serving, swap the scores.

Statistics at 4-hour mark: 608 submissions, 111 accepted, first after 00:11

A — Ace Arbiter

Problem

Check that a sequence of scores could have a appeared as a subsequence in a ping pong game.

Solution

1. We want to know who has what score. If Bob is serving, swap the scores.
2. The serves follow the pattern *ABBAABBAABBAABBA...*, so Bob is serving when $x + y = 1$ or $x + y = 2$ modulo 4.

Statistics at 4-hour mark: 608 submissions, 111 accepted, first after 00:11

A — Ace Arbiter

Problem

Check that a sequence of scores could have a appeared as a subsequence in a ping pong game.

Solution

1. We want to know who has what score. If Bob is serving, swap the scores.
2. The serves follow the pattern *ABBAABBAABBAABBA...*, so Bob is serving when $x + y = 1$ or $x + y = 2$ modulo 4.
3. We now know who has what score at each point in the game. There are only two ways a log can be invalid:
   1. A player's score decreases.
   2. The game continues after someone reaches 11. This includes the tricky case $11 - 11$.

Statistics at 4-hour mark: 608 submissions, 111 accepted, first after 00:11

G — Graduation Guarantee

Problem

For each question in an exam, you can guess the answer correctly with probability p_i. A correct answer gives $+1$ point, an incorrect gives -1. What is the maximum probability to get at least k points?

G — Graduation Guarantee

Problem

For each question in an exam, you can guess the answer correctly with probability p_i. A correct answer gives $+1$ point, an incorrect gives -1. What is the maximum probability to get at least k points?

Solution

1. Sort the list of probabilities. If we decided to answer m questions, then we should pick the m largest p_i.

G — Graduation Guarantee

Problem

For each question in an exam, you can guess the answer correctly with probability p_i. A correct answer gives $+1$ point, an incorrect gives -1. What is the maximum probability to get at least k points?

Solution

1. Sort the list of probabilities. If we decided to answer m questions, then we should pick the m largest p_i.
2. Use dynamic programming to calculate the probability to score at least k points when answering the last m questions, for every m.

G — Graduation Guarantee

Problem

For each question in an exam, you can guess the answer correctly with probability p_i. A correct answer gives $+1$ point, an incorrect gives -1. What is the maximum probability to get at least k points?

Solution

1. Sort the list of probabilities. If we decided to answer m questions, then we should pick the m largest p_i.
2. Use dynamic programming to calculate the probability to score at least k points when answering the last m questions, for every m.
3. $dp(m, q) =$ probability that we get exactly q points when answering the m last questions.

G — Graduation Guarantee

Problem

For each question in an exam, you can guess the answer correctly with probability p_i. A correct answer gives $+1$ point, an incorrect gives -1. What is the maximum probability to get at least k points?

Solution

1. Sort the list of probabilities. If we decided to answer m questions, then we should pick the m largest p_i.
2. Use dynamic programming to calculate the probability to score at least k points when answering the last m questions, for every m.
3. $dp(m, q) =$ probability that we get exactly q points when answering the m last questions.
4. $dp(m, q) = p_m \cdot dp(m-1, q-1) + (1-p_m) \cdot dp(m-1, q+1)$.
5. Time complexity: $\mathcal{O}(nk)$.

B — Berry Battle

Problem

You are given a tree with one berry and one ant in each vertex. When picking a berry, the ants will walk towards that vertex. Pick all the berries so that all ants do not end up in the same vertex.

B — Berry Battle

Problem

You are given a tree with one berry and one ant in each vertex. When picking a berry, the ants will walk towards that vertex. Pick all the berries so that all ants do not end up in the same vertex.

Solution

1. If the tree is a star (diameter ≤ 3) then it is impossible. When the center berry is picked all ants will go to the center. Otherwise, it turns out that it is always possible.

B — Berry Battle

Problem

You are given a tree with one berry and one ant in each vertex. When picking a berry, the ants will walk towards that vertex. Pick all the berries so that all ants do not end up in the same vertex.

Solution

1. If the tree is a star (diameter ≤ 3) then it is impossible. When the center berry is picked all ants will go to the center. Otherwise, it turns out that it is always possible.
2. For ants to stay separated, there has to be two neighbouring ants that never go to the same vertex.

B — Berry Battle

Problem

You are given a tree with one berry and one ant in each vertex. When picking a berry, the ants will walk towards that vertex. Pick all the berries so that all ants do not end up in the same vertex.

Solution

1. If the tree is a star (diameter ≤ 3) then it is impossible. When the center berry is picked all ants will go to the center. Otherwise, it turns out that it is always possible.
2. For ants to stay separated, there has to be two neighbouring ants that never go to the same vertex.
3. **Main idea:** Focus on two neighbouring ants, and try to make them stay separated. There are several ways of doing this.

B — Berry Battle

1. Take two neighbouring vertices u, v that are not leaves.

B — Berry Battle

1. Take two neighbouring vertices u, v that are not leaves.
2. Pick the berry at v. Now there are still ants on both u and v.

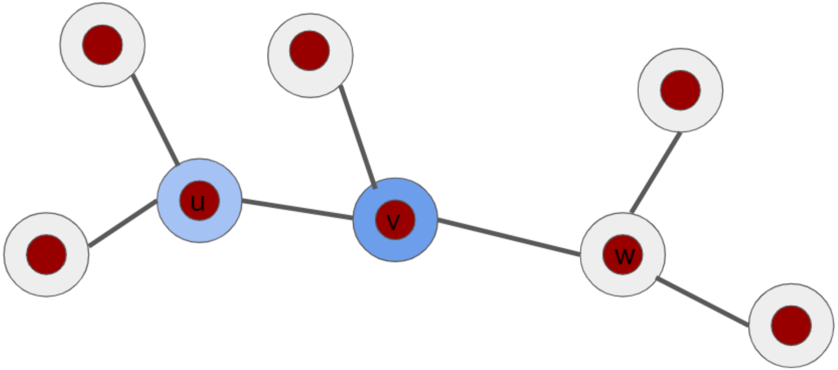
B — Berry Battle

1. Take two neighbouring vertices u, v that are not leaves.
2. Pick the berry at v. Now there are still ants on both u and v.
3. Take a neighbour $w \neq u$ of v, and pick the berry there.

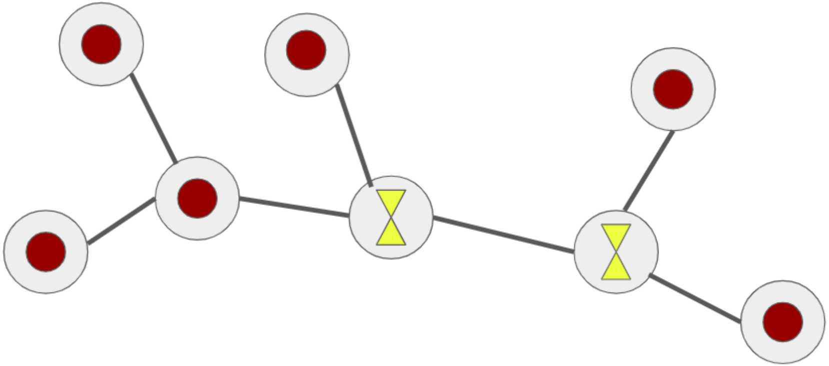
B — Berry Battle

1. Take two neighbouring vertices u, v that are not leaves.
2. Pick the berry at v. Now there are still ants on both u and v.
3. Take a neighbour $w \neq u$ of v, and pick the berry there.
4. Now there are ants on both v and w, but no berries there. In this situation, the remaining berries can be picked in DFS or BFS order.

B — Berry Battle



B — Berry Battle



B — Berry Battle

1. Time complexity: $\mathcal{O}(n)$.
2. Another method is to take the centroid of the tree and alternate between picking in the different subtrees, so that two ants walk back and forth around the centroid.

Statistics at 4-hour mark: 75 submissions, 26 accepted, first after 00:34

J — Junk Journey

Problem

You are a robot on a grid, and your task is to push n scooters into the scooter depot. You are given a generous number of moves.

J — Junk Journey

Problem

You are a robot on a grid, and your task is to push n scooters into the scooter depot. You are given a generous number of moves.

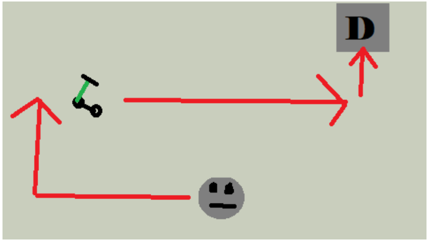
Solution

This is mainly an implementation problem, there are several ways of solving with their own pros and cons. Here are three possible strategies:

J — Junk Journey

Strategy 1: Scooterchasing

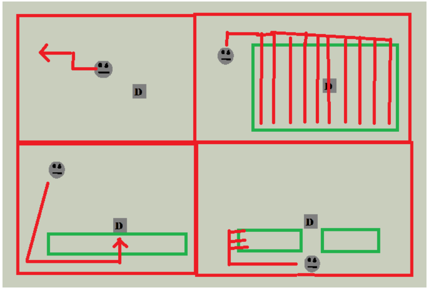
Try to push one scooter in at a time, and ignore the other scooters. You still have to simulate all the movements, but the logic is relatively simple. Pushing one scooter requires roughly 100 moves, so this will probably use around 5000 moves in total.



J — Junk Journey

Strategy 2: Blind Robot

Find a sequence of moves that works regardless of the scooter positions. This is quite tricky, but most of the work can be done on paper. Implementation is very simple, just print the moves.



J — Junk Journey

Strategy 3: Monte Carlo

Try a bunch of random move combinations, greedily pick one that improves the score the most. This is a bit tricky to get to work, but doable.
A good basis for a scoring function is the sum of distances from scooters to the depot. But you also have to make sure the robot doesn't get stuck when far away from scooters.

Statistics at 4-hour mark: 42 submissions, 18 accepted, first after 01:32

F — Foreign Football

Problem

Given all pairwise concatenations of a list of strings, recover the strings.

F — Foreign Football

Problem

Given all pairwise concatenations of a list of strings, recover the strings.

Solution

1. Let $l_1, l_2, \cdots, l_n$ be the lengths of the strings.

F — Foreign Football

Problem

Given all pairwise concatenations of a list of strings, recover the strings.

Solution

1. Let $l_1, l_2, \cdots, l_n$ be the lengths of the strings.
2. $s = l_1 + l_2 + \cdots + l_n$ can be obtained by adding up all lengths in the input.

F — Foreign Football

Problem

Given all pairwise concatenations of a list of strings, recover the strings.

Solution

1. Let $l_1, l_2, \cdots, l_n$ be the lengths of the strings.
2. $s = l_1 + l_2 + \cdots + l_n$ can be obtained by adding up all lengths in the input.
3. $(n-2) \cdot l_i + s$ can be obtained by adding all lengths in row i.

F — Foreign Football

Problem

Given all pairwise concatenations of a list of strings, recover the strings.

Solution

1. Let $l_1, l_2, \cdots, l_n$ be the lengths of the strings.
2. $s = l_1 + l_2 + \cdots + l_n$ can be obtained by adding up all lengths in the input.
3. $(n-2) \cdot l_i + s$ can be obtained by adding all lengths in row i.
4. If $n > 2$, this gives us all lengths l_i. All strings can now easily be obtained.

F — Foreign Football

$n = 2$

New problem: we have two strings a and b, and want to know if there exist strings s, t such that $a = s + t$ and $b = t + s$.

1. Method 1: Try every way of cutting a into $s + t$, use string hashing to check if $b = t + s$.
2. Method 2: Take $b + b$, and check if a is a substring (KMP or hashing).

Statistics at 4-hour mark: 109 submissions, 13 accepted, first after 01:32

E — Enigmatic Enumeration

Problem

Given an undirected graph, count the number of shortest cycles.

Statistics at 4-hour mark: 59 submissions, 12 accepted, first after 00:58

E — Enigmatic Enumeration

Problem

Given an undirected graph, count the number of shortest cycles.

Solution

1. Step 1: Find k, the length of the shortest cycle. For every vertex, find the length of the shortest cycle containing that vertex. This is the distance from the vertex to itself, and can be found with BFS.

Statistics at 4-hour mark: 59 submissions, 12 accepted, first after 00:58

E — Enigmatic Enumeration

Problem

Given an undirected graph, count the number of shortest cycles.

Solution

1. Step 1: Find k, the length of the shortest cycle. For every vertex, find the length of the shortest cycle containing that vertex. This is the distance from the vertex to itself, and can be found with BFS.
2. Step 2: For every vertex v, count the number of shortest cycles going through v. Do a BFS again:
    1. If k is even, for every vertex w of distance $k/2$ from v, count the number of pairs of shortest paths to it.
    2. If k is odd, count the number of edges (u, w) such that u and w are at distance $\frac{k-1}{2}$ from v.

Statistics at 4-hour mark: 59 submissions, 12 accepted, first after 00:58

I — Icy Itinerary

Problem

You are given a graph, and need to find a Hamiltonian path that starts at 1. You can use both edges and non-edges, but can only switch between them at most once.

I — Icy Itinerary

Problem

You are given a graph, and need to find a Hamiltonian path that starts at 1. You can use both edges and non-edges, but can only switch between them at most once.

Solution

1. Start with vertex 1, and insert the other vertices one at a time (in any order).

I — Icy Itinerary

Problem

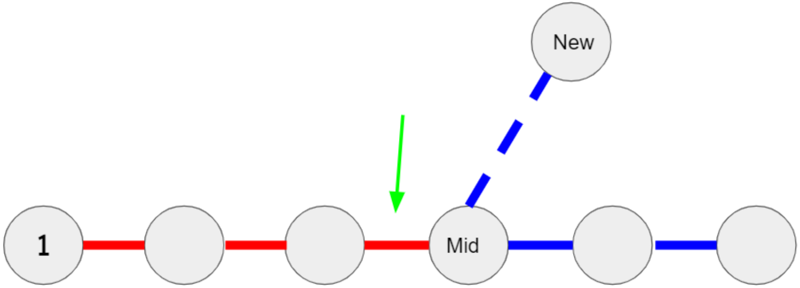
You are given a graph, and need to find a Hamiltonian path that starts at 1. You can use both edges and non-edges, but can only switch between them at most once.

Solution

1. Start with vertex 1, and insert the other vertices one at a time (in any order).
2. **Main observation:** A new vertex can always be inserted in a valid partial path so that the path stays valid!
    1. If the current path only uses edges or non-edges, insert at the end.
    2. Otherwise, it can always be inserted to the left or right of the vertex where the switch happens.

I — Icy Itinerary

Edges are shown in red and non-edges in blue. Look at the color of the edge between **mid** and **new**. Insert **new** on the side of **mid** that has a different color.



Statistics at 4-hour mark: 17 submissions, 2 accepted, first after 01:42

K — Keyboard Queries

Problem

There is an unknown string S. You are given two types of queries: `1 l r` indicates that the substring is a palindrome. `2 x y a b` means that you should answer if the two substrings have to be equal.

K — Keyboard Queries

Problem

There is an unknown string S. You are given two types of queries: `1 l r` indicates that the substring is a palindrome. `2 x y a b` means that you should answer if the two substrings have to be equal.

Solution

1. How to check if two substrings are equal? String hashing!

K — Keyboard Queries

Problem

There is an unknown string S. You are given two types of queries: `1 l r` indicates that the substring is a palindrome. `2 x y a b` means that you should answer if the two substrings have to be equal.

Solution

1. How to check if two substrings are equal? String hashing!
2. Use a segment tree to store the hashes, so that it can handle updates.

K — Keyboard Queries

Problem

There is an unknown string S. You are given two types of queries: `1 l r` indicates that the substring is a palindrome. `2 x y a b` means that you should answer if the two substrings have to be equal.

Solution

1. How to check if two substrings are equal? String hashing!
2. Use a segment tree to store the hashes, so that it can handle updates.
3. When we get a palindrome query, we will learn that some characters are identical.

K — Keyboard Queries

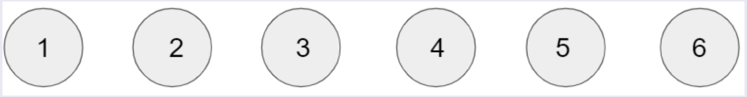
Problem

There is an unknown string S. You are given two types of queries: `1 l r` indicates that the substring is a palindrome. `2 x y a b` means that you should answer if the two substrings have to be equal.

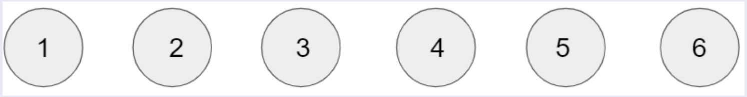
Solution

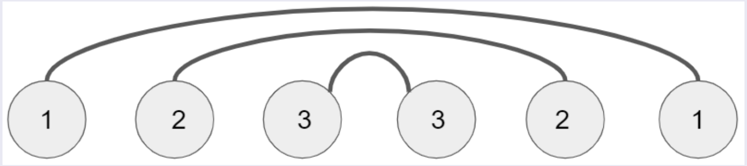
1. How to check if two substrings are equal? String hashing!
2. Use a segment tree to store the hashes, so that it can handle updates.
3. When we get a palindrome query, we will learn that some characters are identical.
4. Use union-find to keep connected components of identical characters. When merging, take the smaller component and change all of its characters.

K — Keyboard Queries



K — Keyboard Queries





K — Keyboard Queries

Solution

1. We don't have time to go through all "edges" in the palindrome queries.

Statistics at 4-hour mark: 29 submissions, 2 accepted, first after 02:11

K — Keyboard Queries

Solution

1. We don't have time to go through all "edges" in the palindrome queries.
2. Use binary search to jump to the point where the merges happen!

Statistics at 4-hour mark: 29 submissions, 2 accepted, first after 02:11

K — Keyboard Queries

Solution

1. We don't have time to go through all "edges" in the palindrome queries.
2. Use binary search to jump to the point where the merges happen!
3. There are only $\leq n - 1$ merges in total.

Statistics at 4-hour mark: 29 submissions, 2 accepted, first after 02:11

K — Keyboard Queries

Solution

1. We don't have time to go through all "edges" in the palindrome queries.
2. Use binary search to jump to the point where the merges happen!
3. There are only $\leq n - 1$ merges in total.
4. Another optimization: When getting a query of type 2, start by checking if it is already a palindrome.

Statistics at 4-hour mark: 29 submissions, 2 accepted, first after 02:11

K — Keyboard Queries

Solution

1. We don't have time to go through all "edges" in the palindrome queries.
2. Use binary search to jump to the point where the merges happen!
3. There are only $\leq n-1$ merges in total.
4. Another optimization: When getting a query of type 2, start by checking if it is already a palindrome.
5. Time complexity: $\mathcal{O}(n \log^2 n + q \log n)$.

Statistics at 4-hour mark: 29 submissions, 2 accepted, first after 02:11

Results!